

# *HPSS Programmer's Reference*

*High Performance Storage System  
Release 6.2*

*July 2008 (Revision 2.0)*

© Copyright (C) 1992, 2008 International Business Machines Corporation, The Regents of the University of California, Los Alamos National Security, LLC, Lawrence Livermore National Security, LLC, Sandia Corporation, and UT-Battelle.

All rights reserved.

Portions of this work were produced by Lawrence Livermore National Security, LLC, Lawrence Livermore National Laboratory (LLNL) under Contract No. DE-AC52-07NA27344 with the U.S. Department of Energy (DOE); by the University of California, Lawrence Berkeley National Laboratory (LBNL) under Contract No. DE-AC02-05CH11231 with DOE; by Los Alamos National Security, LLC, Los Alamos National Laboratory (LANL) under Contract No. DE-AC52-06NA25396 with DOE; by Sandia Corporation, Sandia National Laboratories (SNL) under Contract No. DE-AC04-94AL85000 with DOE; and by UT-Battelle, Oak Ridge National Laboratory (ORNL) under Contract No. DE-AC05-00OR22725 with DOE. The U.S. Government has certain reserved rights under its prime contracts with the Laboratories.

#### DISCLAIMER

Portions of this software were sponsored by an agency of the United States Government. Neither the United States, DOE, The Regents of the University of California, Los Alamos National Security, LLC, Lawrence Livermore National Security, LLC, Sandia Corporation, UT-Battelle, nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America.

HPSS Release 6.2

July 2008 (Revision 2.0)

High Performance Storage System is a trademark of International Business Machines Corporation.

IBM is a registered trademark of International Business Machines Corporation.

IBM, DB2, DB2 Universal Database, AIX, pSeries, and xSeries are trademarks or registered trademarks of International Business Machines Corporation.

AIX and RISC/6000 are trademarks of International Business Machines Corporation.

UNIX is a registered trademark of the Open Group.

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

Kerberos is a trademark of the Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Incorporated in the United States and other countries.

ACSL is a trademark of Sun Microsystems, Incorporated.

Microsoft Windows is a registered trademark of Microsoft Corporation.

NFS, Network File System, and ACSL are trademarks of Sun Microsystems, Inc.

DST is a trademark of Ampex Systems Corporation.

Other brands and product names appearing herein may be trademarks or registered trademarks of third parties.

# Preface

This *High Performance Storage System (HPSS) Programmer's Reference Guide, Release 6.2*, documents client function calls which are provided by HPSS. It is designed for application programmers.

The document provides the programming reference for HPSS Release 6.2. In particular, the HPSS Client Application Programming Interfaces (APIs), 64-bit arithmetic library calls, Site Interfaces, ACL Interfaces, Configuration Parser Interfaces, and Real Time Monitoring System Interfaces are described. The 64-bit arithmetic library APIs are included since some Client APIs require 64-bit unsigned integer fields.

The objective of this document is to meet the following general goals:

- Define any known limitations of the APIs.
- Define the application programming interfaces (APIs) provided for use by other subsystems or clients.
- Define the data definitions referenced by the APIs.

Starting with Release 6.2, the legacy `hpss_ReadList` and `hpss_WriteList` functions are being replaced with the new HPSS Parallel I/O (PIO) functions. Therefore the legacy information has been removed from this document to the *HPSS Programmer's Reference - I/O Supplement* document. For sites requiring information on the legacy functions, please contact your Support Representative to obtain the supplemental document.

Refer to the *HPSS User's Guide* for a description of the following command line interfaces: standard FTP, parallel FTP, HPSS File System client and user utilities.

Refer to the *HPSS Error Messages Manual* for a list of all HPSS error and advisory messages which are output by the HPSS software. For each message, the following information is provided: message identifier and text, source file name(s) which generate the message, problem description, system action, and administrator action.

Refer to the *HPSS Installation Guide* and *HPSS Management Guide* for descriptions of the interfaces provided to HPSS administrators.

The *HPSS Programmer's Reference* is structured as follows:

## **Overview**

This chapter provides an overview of each type of programmer interface, constraints, and required libraries.

## **Client API**

This chapter defines the Client API specifications and associated data definitions.

## **Math Library**

This chapter defines the 64-bit arithmetic library API specifications and associated data definitions.

## **Site Interfaces**

This chapter defines the Account Validation and Gatekeeper Site Interface specifications and associated data definitions.

## **ACL API**

This chapter defines the specifications and associated data definitions for the Access Control List (ACL) API.

## **Configuration Parser**

This chapter defines the specifications and associated data definitions for the Configuration Parser Interface.

### ***Real Time Monitoring***

This chapter defines the specifications and associated data definitions for the Real Time Monitoring Service.

### ***Glossary of Terms and Acronyms***

This appendix provides a list of HPSS related terms and acronyms used in this document.

### ***References***

This appendix lists documents cited in the text as well as other reference materials.

### ***Acknowledgments***

This appendix acknowledges contributions to the HPSS development.

### ***Typographic and Keying Conventions***

This document uses the following typographic conventions:

**Bold**    **Bold** words or characters represent system elements to be used literally, such as functions, commands or keywords.

*Italic*    *Italic* words or characters represent variable values to be supplied.

[ ]        Brackets enclose optional items in syntax and format descriptions.

{ }        Braces enclose a list of items to select in syntax and format descriptions.

# Table of Contents

<b>Chapter 1. Overview .....</b>	<b>15</b>
1.1.Client API .....	15
1.1.1.Purpose.....	15
1.1.2.Components.....	15
1.1.3.Constraints.....	17
1.1.4.Libraries.....	17
1.1.5.Environment Variables.....	17
1.2.Network Options.....	19
1.3.64-bit Arithmetic Library.....	19
1.3.1.Purpose.....	19
1.3.2.Components.....	19
1.3.3.Constraints.....	20
1.3.4.Libraries.....	20
1.4.Storage Concepts .....	20
1.4.1.Class of Service.....	21
1.4.2.Storage Class.....	21
1.4.3.Storage Hierarchy.....	21
1.4.4.File Family.....	21
1.5.User Ids .....	22
1.6.Access Control List API.....	22
1.6.1.Purpose.....	22
1.6.2.Components.....	22
1.6.3.Constraints.....	23
1.6.4.Libraries.....	23
1.6.5.Purpose.....	23
1.6.6.Components.....	23
1.6.7.Constraints.....	24
1.6.8.Libraries.....	24
<b>Chapter 2. Client API Functions.....</b>	<b>27</b>
2.1.API Interfaces.....	27
2.1.1.hpss_Access.....	27
2.1.2.hpss_AccessHandle.....	28
2.1.3.hpss_AcctCodeToName.....	30
2.1.4.hpss_AcctNameToCode.....	30
2.1.5.hpss_BeginExTrans.....	31
2.1.6.hpss_Chacct.....	32
2.1.7.hpss_ChacctByName.....	33
2.1.8.hpss_Chdir.....	34
2.1.9.hpss_Chmod.....	35

2.1.10.hpss_Chown.....	35
2.1.11.hpss_Chroot.....	36
2.1.12.hpss_ClientAPIInit.....	37
2.1.13.hpss_ClientAPIReset.....	38
2.1.14.hpss_Close.....	39
2.1.15.hpss_Closedir.....	39
2.1.16.hpss_ConvertIdsToNames.....	40
2.1.17.hpss_ConvertNamesToIds.....	41
2.1.18.hpss_CopyFile.....	43
2.1.19.hpss_Creat.....	43
2.1.20.hpss_Create .....	44
2.1.21.hpss_CreateDMHandle.....	46
2.1.22.hpss_CreateHandle.....	47
2.1.23.hpss_CreateWithHints.....	49
2.1.24.hpss_DeleteACL.....	50
2.1.25.hpss_DeleteACLHandle.....	51
2.1.26.hpss_EndExTrans.....	52
2.1.27.hpss_Fclear.....	53
2.1.28.hpss_FclearOffset.....	54
2.1.29.hpss_Fclose.....	55
2.1.30.hpss_Fcntl.....	56
2.1.31.hpss_Fflush.....	57
2.1.32.hpss_Fgetc.....	57
2.1.33.hpss_Fgets.....	58
The behavior of this function is modeled after the system fgets() routine.....	59
2.1.34.hpss_FileGetAttributes.....	59
2.1.35.hpss_FileGetAttributesHandle.....	60
2.1.36.hpss_FileGetAttributesSOID.....	61
2.1.37.hpss_FileGetXAttributes.....	62
2.1.38.hpss_FileSetAttributes.....	64
2.1.39.hpss_FileSetAttributesHandle.....	66
2.1.40.hpss_FileSetAttributesSOID.....	67
2.1.41.hpss_FilesetCreate.....	68
2.1.42.hpss_FilesetDelete.....	70
2.1.43.hpss_FilesetGetAttributes.....	71
2.1.44.hpss_FilesetListAll.....	72
2.1.45.hpss_FilesetSetAttributes.....	73
2.1.46.hpss_Fopen.....	74
2.1.47.hpss_Fpreallocate.....	75
2.1.48.hpss_Fread.....	76
2.1.49.hpss_Fseek.....	77
2.1.50.hpss_Fstat.....	78
2.1.51.hpss_Fsync.....	79
2.1.52.hpss_Ftell.....	80

2.1.53.hpss_Ftruncate.....	80
2.1.54.hpss_Fwrite.....	81
2.1.55.hpss_GetAcct.....	82
2.1.56.hpss_GetAcctName.....	83
2.1.57.hpss_GetACL.....	84
2.1.58.hpss_GetACLHandle.....	85
2.1.59.hpss_GetAsynchStatus.....	86
2.1.60.hpss_GetAttrHandle.....	87
2.1.61.hpss_GetAuthType.....	88
2.1.62.hpss_GetConfiguration.....	89
2.1.63.hpss_Getcwd.....	89
2.1.64.hpss_GetDistFile.....	90
2.1.65.hpss_GetJunctions.....	91
2.1.66.hpss_GetListAttrs.....	92
2.1.67.hpss_GetJunctionAttrs.....	93
2.1.68.hpss_GetObjId.....	94
2.1.69.hpss_GetObjType.....	95
2.1.70.hpss_GetPathHandle.....	96
2.1.71.hpss_GetRawAttrHandle.....	96
2.1.72.hpss_GetSubSysStats.....	97
2.1.73.hpss_GetThreadUcred.....	98
2.1.74.hpss_HandleCompare.....	99
2.1.75.hpss_InsertDistFile.....	100
2.1.76.hpss_JunctionCreate.....	101
2.1.77.hpss_JunctionCreateHandle.....	102
2.1.78.hpss_JunctionDelete.....	103
2.1.79.hpss_JunctionDeleteHandle.....	103
2.1.80.hpss_Link.....	104
2.1.81.hpss_LinkHandle.....	105
2.1.82.hpss_LoadThreadState.....	106
2.1.83.hpss_LoadDefaultThreadState.....	107
2.1.84.hpss_Lseek.....	108
2.1.85.hpss_Lstat.....	109
2.1.86.hpss_Migrate.....	110
2.1.87.hpss_Mkdir.....	111
2.1.88.hpss_MkdirHandle.....	112
2.1.89.hpss_Open.....	113
2.1.90.hpss_OpenBitfile.....	115
2.1.91.hpss_OpenBitfileVAttrs.....	116
2.1.92.hpss_OpenHandle.....	118
2.1.93.hpss_Opendir.....	120
2.1.94.hpss_OpenWithHints.....	121
2.1.95.hpss_PIOEnd.....	122
2.1.96.hpss_PIOExecute.....	122

2.1.97.hpss_PIOExportGrp.....	123
2.1.98.hpss_PIOImportGrp.....	124
2.1.99.hpss_PIORegister.....	125
2.1.100.hpss_PIOStart.....	126
2.1.101.hpss_Purge.....	126
2.1.102.hpss_PurgeLock.....	127
2.1.103.hpss_PurgeLoginCred.....	128
2.1.104.hpss_Read.....	129
2.1.105.hpss_ReadAttrs.....	130
2.1.106.hpss_ReadAttrsHandle.....	131
2.1.107.hpss_Readdir.....	132
2.1.108.hpss_ReaddirHandle.....	133
2.1.109.hpss_Readlink.....	134
2.1.110.hpss_ReadlinkHandle.....	135
2.1.111.hpss_ReadRawAttrsHandle.....	136
2.1.112.hpss_Rename.....	138
2.1.113.hpss_RenameHandle.....	139
2.1.114.hpss_ReopenBitfile.....	140
2.1.115.hpss_ResetSubSysStats.....	141
2.1.116.hpss_Rewinddir.....	142
2.1.117.hpss_Rmdir.....	143
2.1.118.hpss_RmdirHandle.....	144
2.1.119.hpss_RmdirNoLookup.....	145
2.1.120.hpss_SetACL.....	146
2.1.121.hpss_SetACLHandle.....	147
2.1.122.hpss_SetAcct.....	148
2.1.123.hpss_SetAcctByName.....	149
2.1.124.hpss_SetAttrHandle.....	150
2.1.125.hpss_SetAuditInfo.....	151
2.1.126.hpss_SetConfiguration.....	152
2.1.127.hpss_SetCOSByHints.....	153
2.1.128.hpss_SetFileOffset.....	154
2.1.129.hpss_SetLoginCred.....	155
2.1.130.hpss_SiteIdToName.....	156
2.1.131.hpss_SiteNameToId.....	157
2.1.132.hpss_Stage.....	158
2.1.133.hpss_StageCallBack.....	159
2.1.134.hpss_Stat.....	160
2.1.135.hpss_Statfs.....	161
2.1.136.hpss_Statvfs.....	162
2.1.137.hpss_Symlink.....	163
2.1.138.hpss_SymlinkHandle.....	163
2.1.139.hpss_ThreadCleanUp.....	165
2.1.140.hpss_Truncate.....	165



2.1.141.hpss_TruncateHandle.....	166
2.1.142.hpss_Umask.....	167
2.1.143.hpss_Unlink.....	168
2.1.144.hpss_UnlinkHandle.....	169
2.1.145.hpss_UnlinkNoLookup.....	170
2.1.146.hpss_UpdateACL.....	171
2.1.147.hpss_UpdateACLHandle.....	173
2.1.148.hpss_Utime.....	174
2.1.149.hpss_Write.....	175
2.2.Data Definitions.....	176
2.2.1.Account Record - acct_rec_t.....	177
2.2.2.API Configuration Structure – api_config_t.....	177
2.2.3.API Distributed File Information – api_dist_file_info_t.....	179
2.2.4.API DMAP Attributes – api_dmap_attr_t.....	180
2.2.5.API Name Specification – api_namespec_t.....	181
2.2.6.Bitfile Volatile and Metadata Attributes - bf_attr_t.....	182
2.2.7.Bitfile Volatile and Metadata Extended Attributes - bf_xattr_t.....	182
2.2.8.Bitfile Metadata Attributes - bf_attr_md_t.....	183
2.2.9.Bitfile Service Storage Class Attributes - bf_sc_attr_t.....	185
2.2.10.Bitfile Service Virtual Volume Attributes - bf_vv_attr_t.....	186
2.2.11.Bitfile Callback Address – bfs_callback_addr_t.....	187
2.2.12.Core Server Attribute Structure - hpss_Attrs_t.....	188
2.2.13.Name Service Object Attribute Bits – hpss_AttrBits_t.....	192
2.2.14.Authentication Mechanism Type – hpss_authn_mech_t.....	192
2.2.15.Authentication Token Type – hpss_authz_token_t.....	193
2.2.16.File Creation Hint Structure - hpss_cos_hints_t.....	194
2.2.17.Class of Service Priorities - hpss_cos_priorities_t.....	196
2.2.18.Class of Service Metadata Structure - hpss_cos_md_t.....	197
2.2.19.HPSS Directory Entry – hpss_dirent_t.....	199
2.2.20.Extended Transaction Outcome – hpss_ExTrans_Outcome_t.....	200
2.2.21.File Attribute Structure - hpss_fileattr_t.....	200
2.2.22.File Attributes Bit Data Type – hpss_fileattrbits_t.....	200
2.2.23.Extended File Attribute Structure - hpss_xfileattr_t.....	201
2.2.24.Global Fileset Entry Structure – hpss_global_fsentry_t.....	201
2.2.25.Parallel I/O Callback Type – hpss_pio_cb_t.....	202
2.2.26.Parallel I/O Gap Information Type – hpss_pio_gapinfo_t.....	202
2.2.27.Parallel Stripe Group Type – hpss_pio_grp_t.....	202
2.2.28.HPSS Parallel I/O Operation Type – hpss_pio_operation_t.....	203
2.2.29.HPSS Parallel I/O Parameters – hpss_pio_params_t.....	203
2.2.30.HPSS Parallel I/O Transport Type – hpss_pio_transport_t.....	204
2.2.31.HPSS I/O Request ID Type – hpss_reqid_t.....	204
2.2.32.HPSS RPC Protection Level Type – hpss_rpc_prot_level_t.....	204
2.2.33.HPSS File Statistics Data Structure – hpss_stat_t.....	204

2.2.34.HPSS VFS File Statistics Structure – hpss_statvfs_t.....	207
2.2.35.HPSS Name Space Attributes Structure – hpss_vattr_t.....	208
2.2.36.HPSS UUID Type – hpss_uuid_t.....	210
2.2.37.HPSS Namespec Type – namespec_type_t.....	211
2.2.38.Name Service ACL Conformant Array - ns_ACLConfArray_t.....	211
2.2.39.Name Service Access Control List Entry - ns_ACLEntry_t.....	211
2.2.40.Name Service Directory Entry - ns_DirEntry_t.....	212
2.2.41.Name Service Fileset Attributes Structure – ns_FilesetAttr_t.....	213
2.2.42.Name Service Fileset Attribute Bits – ns_FilesetAttrBits_t.....	215
2.2.43.Name Service Object Handle Structure - ns_ObjHandle_t.....	215
2.2.44.HPSS Object Handle Structure – hpss_object_handle_t.....	216
2.2.45.Purge Lock Flag - purgelock_flag_t.....	216
2.2.46.Core Server Physical Volume Attributes - pv_list_element_t.....	217
2.2.47.Core Server Physical Volume Attributes Conformant Array - pv_list_t.....	217
2.2.48.HPSS Security User Credentials – sec_cred_t.....	218
2.2.49.Subsystem Statistics - subsys_stats_t.....	219
2.2.50.Timestamp Seconds Type – timestamp_sec_t.....	220
<b>2.3.Network Options .....</b>	<b>220</b>
2.3.1.Network Options Functions.....	220
2.3.1.1.netopt_FindEntry.....	220
2.3.1.2.netopt_GetWriteSize.....	221
2.3.2.Network Options Data Definitions.....	222
2.3.2.1.Network Options Entry - netopt_entry_t.....	222
<b>Chapter 3. Math Library.....</b>	<b>225</b>
3.1.1.add64m.....	225
3.1.2.add64_3m.....	226
3.1.3.and64m.....	226
3.1.4.bld64m.....	227
3.1.5.cast64m.....	228
3.1.6.div64m.....	228
3.1.7.div2x64m.....	229
3.1.8.div_cl64m.....	230
3.1.9.div_2xcl64m.....	230
3.1.10.eqz64m.....	231
3.1.11.eq64m.....	232
3.1.12.ge64m.....	232
3.1.13.gt64m.....	233
3.1.14.high32m.....	234
3.1.15.le64m.....	234
3.1.16.low32m.....	235
3.1.17.lt64m.....	236
3.1.18.mod64m.....	236
3.1.19.mod2x64m.....	237

3.1.20.mul64m.....	238
3.1.21.neqz64m.....	238
3.1.22.not64m.....	239
3.1.23.or64m.....	240
3.1.24.shl64m.....	240
3.1.25.shr64m.....	241
3.1.26.sub64m.....	242
3.1.27.sub64_3m.....	242
<b>3.2.Data Definitions.....</b>	<b>243</b>
3.2.1.u_signed64.....	243
3.2.2.unsigned32.....	244
<b>Chapter 4. Site Interfaces.....</b>	<b>245</b>
4.1.Gatekeeping.....	245
4.1.1.gk_site_Close.....	245
4.1.2.gk_site_Create.....	247
4.1.3.gk_site_CreateComplete.....	251
4.1.4.gk_site_CreateStats.....	252
4.1.5.gk_site_GetMonitorTypes.....	254
4.1.6.gk_site_Init.....	256
4.1.7.gk_site_Open.....	257
4.1.8.gk_site_OpenStats.....	259
4.1.9.gk_site_PassThru.....	260
4.1.10.gk_site_ReadSitePolicy.....	261
4.1.11.gk_site_Shutdown.....	263
4.1.12.gk_site_Stage.....	264
4.1.13.gk_site_StageComplete.....	265
4.1.14.gk_site_StageStats.....	266
4.2.Account Validation Site Interface.....	267
4.2.1.av_site_AcctIdxToName.....	267
4.2.2.av_site_AcctNameToIdx.....	269
4.2.3.av_site_Initialize.....	272
4.2.4.av_site_Shutdown.....	273
4.2.5.av_site_ValidateAccount.....	274
4.2.6.av_site_ValidateChacct.....	276
4.2.7.av_site_ValidateChown.....	278
4.2.8.av_site_ValidateCreate.....	280
<b>Chapter 5. Access Control List API Functions.....</b>	<b>283</b>
5.1.API Interfaces.....	283
5.1.1.hacl_ConvertACLToHACL.....	283
5.1.2.hacl_ConvertHACLToACL.....	284
5.1.3.hacl_ConvertHACLToString.....	284

5.1.4.hacl_ConvertHACLPermsToPerms.....	286
5.1.5.hacl_ConvertHACLTypeToType.....	286
5.1.6.hacl_ConvertPermsToHACLPerms.....	287
5.1.7.hacl_ConvertStringsToHACL.....	288
5.1.8.hacl_ConvertTypeToHACLType.....	289
5.1.9.hacl_DeleteHACL.....	290
5.1.10.hacl_GetHACL.....	291
5.1.11.hacl_SetHACL.....	292
5.1.12.hacl_SortHACL.....	293
5.1.13.hacl_UpdateHACL.....	293
5.2.Data Definitions.....	295
5.2.1.HACL-style Access Control List - hacl_acl_t.....	295
5.2.2.HACL-style Access Control List Entry - hacl_acl_entry_t.....	295
<b>Chapter 6. HPSS File System Client Interfaces.....</b>	<b>297</b>
6.1.1.HPSSFS_GET_COS.....	297
6.1.2.HPSSFS_SET_COS_HINT.....	298
6.1.3.HPSSFS_SET_FSIZE_HINT.....	299
6.1.4.HPSSFS_SET_MAXSEGSZ_HINT.....	300
6.1.5.HPSSFS_PURGE_CACHE.....	301
<b>Chapter 7. Configuration Parser Functions.....</b>	<b>303</b>
7.1.1.hpsscfx_ConfGetClientInterfaces.....	303
7.1.2.hpss_CfgParse.....	304
7.1.3.hpss_CfgFindToken.....	305
7.1.4.hpss_CfgFindKey.....	306
7.1.5.hpssFree.....	306
7.1.6.hpsscfx_LookupHostName.....	307
7.1.7.hpsscfx_ConfParse.....	308
7.1.8.hpsscfx_ConfSetPathDirs.....	309
7.1.9.hpsscfx_ConfSetFileName.....	310
7.1.10.hpsscfx_ConfSetDirPaths.....	310
7.1.11.hpsscfx_NetoptFindEntry.....	311
7.1.12.hpsscfx_NetoptSetSock.....	312
7.1.13.hpsscfx_NetoptGetWriteSize.....	313
7.1.14.hpsscfx_PatternMatch.....	313
7.1.15.hpsscfx_GetRestrictedPorts.....	314
7.1.16.ai_thread_abort.....	315
7.1.17.ai_thread_detach.....	316
7.1.18.ai_thread_exit.....	316
7.1.19.ai_thread_create.....	317
7.1.20.ai_thread_equal.....	318
7.1.21.ai_thread_join.....	318
7.1.22.ai_thread_self.....	319

7.1.23.ai_thread_sleep.....	320
7.1.24.ai_thread_yield.....	320
7.1.25.ai_mutex_destroy.....	321
7.1.26.ai_mutex_init.....	322
7.1.27.ai_mutex_lock.....	322
7.1.28.ai_mutex_try_lock.....	323
7.1.29.ai_mutex_unlock.....	324
7.1.30.ai_condition_broadcast.....	324
7.1.31.ai_condition_destroy.....	325
7.1.32.ai_condition_init.....	326
7.1.33.ai_condition_signal.....	326
7.1.34.ai_condition_wait.....	327
7.1.35.ai_condition_timedwait.....	328
7.2.Data Definitions.....	329
7.2.1.Defines.....	329
7.2.2.hpss_cfg_stanza_t.....	330
<b>Chapter 8. Real Time Monitoring .....</b>	<b>333</b>
8.1.RTM API Interfaces.....	333
8.1.1.rtm_GetRequestEntries.....	333
8.2.RTM API Data Definitions.....	335
8.2.1.RTM Object Type Enumeration - rtm_objectname_t.....	335
8.2.2.RTM Object Type - rtm_object_t.....	335
8.2.3.Request Array - rtm_req_array_t.....	336
8.2.4.Request Entry - rtm_req_array_entry_t.....	336
8.2.5.Request Code Enumeration - rtm_request_code_t.....	338
8.2.6.Request State - rtm_state_t.....	340
8.2.7.Server Request Information - rtm_serverspecific_t.....	340
8.2.8.Wait Reason Enumeration - rtm_wait_reason_t.....	341
8.2.9.Wait Resource - rtm_wait_resource_t.....	343
8.2.10.Wait Type Enumeration - rtm_wait_type_t.....	344
8.2.11.CORE Request - rtm_core_req_attr_t.....	345
8.2.12.Mover Request - rtm_mvr_req_attr_t.....	348
8.2.13.Gatekeeper Request - rtm_gk_req_attr_t.....	349
8.3.RTM Library Routines.....	352
8.3.1.rtm_Init.....	352
8.3.2.rtm_Shutdown.....	353
8.3.3.rtm_ReqListInit.....	353
8.3.4.rtm_ReqListInsertEntry.....	354
8.3.5.rtm_ReqListDeleteEntry.....	355
8.3.6.rtm_ReqListUpdateEntry.....	356
8.3.7.rtm_WaitListInsertEntry.....	357
8.3.8.rtm_WaitListDeleteEntry.....	358

8.3.9.rtm_WaitListUpdateEntry.....	359
Appendix A. Glossary of Terms and Acronyms.....	361
Appendix B. References.....	374
Appendix C. Developer Acknowledgments.....	376

# Chapter 1. Overview

The High Performance Storage System (HPSS) provides scalable parallel storage systems for highly parallel computers as well as traditional supercomputers and workstation clusters. Concentrating on meeting the high end of storage system and data management requirements, HPSS is scalable and designed for large storage capacities, and to use network-connected storage devices to transfer data at rates up to multiple gigabytes per second. Listed below are the programming interfaces for accessing data from HPSS.

## 1.1. Client API

### 1.1.1. Purpose

The purpose of the Client API is to provide an interface which mirrors the POSIX.1 specification where possible to provide ease of use to the POSIX application programmer. In addition, extensions to allow the programmer to take advantage of the specific features provided by HPSS are provided (e.g., storage/access hints passed on file creation, parallel data transfers, migration, and purge).

### 1.1.2. Components

The Client API consists of these major parts:

- File Open/Creation and Close Operations
- File Data Access Operations
- Fileset/Junction Creation and Deletion Operations
- File Attribute Operations
- File Name Operations
- Directory Creation and Deletion Operations
- Directory Access Operations
- Working Directory Operations
- Client API Control Operations
- Security Context Routines
- Core Server Statistic Operations
- HPSS Object Manipulation Routines
- Streams File Operations

File Open/Creation and Close Operations provide functions to create a file, open existing files and close previously opened files. The functions within this section are **hpss\_Open**, **hpss\_OpenHandle**, **hpss\_Close**, **hpss\_Creat**, **hpss\_Create**, **hpss\_CreateHandle**, **hpss\_CreateDMHandle**, **hpss\_OpenBitfile**, **hpss\_OpenBitfileVAttr**, **hpss\_SetCOSByHints**, and **hpss\_ReopenBitfile**.

File Data Access Operations provide functions to read from and write data to HPSS files. The functions within this section include **hpss\_Lseek**, **hpss\_Read**, **hpss\_ReadList**, **hpss\_SetFileOffset**, **hpss\_GetDistFile**, **hpss\_InsertDistFile**, **hpss\_CopyFile**, **hpss\_Fpreallocate**, **hpss\_Write**, **hpss\_WriteList**, **hpss\_BeginExTrans**, **hpss\_EndExTrans**, and **hpss\_GetAsynchStatus**.

Fileset/Junction Creation and Deletion Operation provide functions to create and delete filesets and junctions. Functions within this section include **hpss\_FilesetCreate**, **hpss\_FilesetDelete**, **hpss\_FilesetGetAttributes**, **hpss\_FilesetSetAttributes**, **hpss\_FilesetListAll**, **hpss\_GetJunctions**, **hpss\_JunctionCreate**, **hpss\_JunctionCreateHandle**, **hpss\_JunctionDelete**, and **hpss\_JunctionDeleteHandle**.

File Attribute Operations include functions to query and alter a file's attribute values (both via POSIX consistent interfaces and extended HPSS interfaces), and determine a client's accessibility to a file or directory. Functions within this section are **hpss\_Access**, **hpss\_AccessHandle**, **hpss\_AcctCodeToName**, **hpss\_AcctNameToCode**, **hpss\_Chacct**, **hpss\_ChacctByName**, **hpss\_Chmod**, **hpss\_Chown**, **hpss\_ConvertIdsToNames**, **hpss\_ConvertNamesToIds**, **hpss\_DeleteACL**, **hpss\_DeleteACLHandle**, **hpss\_Fclear**, **hpss\_FclearOffset**, **hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_GetAttrHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileGetXAttributes**, **hpss\_GetRawAttrHandle**, **hpss\_FileSetAttributes**, **hpss\_FileSetAttributesHandle**, **hpss\_FileSetAttributesSOID**, **hpss\_Fstat**, **hpss\_Ftruncate**, **hpss\_GetAcct**, **hpss\_GetAcctName**, **hpss\_GetACL**, **hpss\_GetACLHandle**, **hpss\_GetListAttrs**, **hpss\_GetJunctionAttrs**, **hpss\_Lstat**, **hpss\_Migrate**, **hpss\_Purge**, **hpss\_SetACL**, **hpss\_SetACLHandle**, **hpss\_SetAcct**, **hpss\_SetAcctByName**, **hpss\_SiteIdToName**, **hpss\_SiteNameToId**, **hpss\_Stage**, **hpss\_StageCallBack**, **hpss\_Stat**, **hpss\_Statfs**, **hpss\_Statvfs**, **hpss\_Truncate**, **hpss\_TruncateHandle**, **hpss\_Umask**, **hpss\_UpdateACL**, **hpss\_UpdateACLHandle**, **hpss\_Utime**, and **hpss\_PurgeLock**.

File Name Operations are used to rename files and directories and remove a name associated with a file. Functions within this section include **hpss\_Link**, **hpss\_LinkHandle**, **hpss\_Readlink**, **hpss\_ReadlinkHandle**, **hpss\_Rename**, **hpss\_RenameHandle**, **hpss\_Symlink**, **hpss\_SymlinkHandle**, **hpss\_Unlink**, **hpss\_UnlinkHandle**, and **hpss\_UnlinkNoLookup**.

Directory Creation and Deletion Operations provide functions to make and remove directories. Functions within this section include **hpss\_Mkdir**, **hpss\_MkdirHandle**, **hpss\_Rmdir**, **hpss\_SetAttrHandle**, **hpss\_RmdirHandle**, and **hpss\_RmdirNoLookup**.

Directory Access Operations provide functions to read the directory entries from a directory. Functions within this section include **hpss\_Closedir**, **hpss\_Opendir**, **hpss\_ReadAttrs**, **hpss\_ReadAttrsHandle**, **hpss\_ReadRawAttrsHandle**, **hpss\_Readdir**, **hpss\_ReaddirHandle**, and **hpss\_Rewinddir**.

Working Directory Operations provide functions to query and alter a thread's current working directory. Functions within this section include **hpss\_Chdir**, **hpss\_Chroot**, and **hpss\_Getcwd**.

The Client API Control Operations, to update and clean up a thread's Client API state information, within this section include **hpss\_GetConfiguration**, **hpss\_LoadThreadState**, **hpss\_LoadDefaultThreadState**, **hpss\_ThreadCleanUp**, and **hpss\_SetConfiguration**. Also included are two important internal routines: **hpss\_ClientAPIInit**, and **hpss\_ClientAPIReset**. These APIs will not be used by most applications.

Security Context Operations provide convenience functions to establish an application program's login context, and subsequently purge the login context. An application program which is calling the Client API library must run on behalf of either a Unix or Kerberos principal. Either the user can acquire security credentials prior to submitting the application program, or the **hpss\_SetLoginCred** function may be called from the application. The name of the principal and associated keytab file name are supplied to the function. Prior to exiting the application, the user must call **hpss\_PurgeLoginCred** to delete the security context and terminate the thread which maintains the context. The user credentials, that the currently running thread has, can be obtained using the routine **hpss\_GetThreadUcred**. Security audit information can be obtained on a per thread basis using the routine **hpss\_SetAuditInfo**.

Core Server statistics operations provide the ability to get and reset the stage, migration, purge, and delete counts for the subsystem. The functions that provide these capabilities are **hpss\_GetSubSysStats** and **hpss\_ResetSubSysStats**.

HPSS Object manipulation routines are **hpss\_GetObjId**, **hpss\_GetObjType**, **hpss\_GetPathHandle**, and **hpss\_HandleCompare**.



Buffered File Operations provide a "stream-like" interface for buffered I/O operations between the client and HPSS. These functions are: **hpss\_Fclose**, **hpss\_Fcntl**, **hpss\_Fflush**, **hpss\_Fopen**, **hpss\_Fread**, **hpss\_Fseek**, **hpss\_Fstat**, **hpss\_Fsync**, **hpss\_Ftell**, and **hpss\_Fwrite**.

Additional functions are provided to allow COS hints: **hpss\_CreateWithHints** and **hpss\_OpenWithHints**.

### 1.1.3. Constraints

The following constraints are being imposed by the Client API:

- The validity of open files and directories at the time of **fork** is undefined in the child process.
- The validity of open files and directories is lost across calls to any of the family of **exec** calls.
- The designed client API works only with applications that make use of the Core Server. In particular, this API is not designed to meet the needs of clients that will perform the Name Service functionality internally and/or will bypass the Core Server when performing storage operations.

### 1.1.4. Libraries

Applications issuing HPSS Client API calls must link with the following library:

- **libhpss.a** HPSS client library

Applications using the Posix compliant functions must link with the following library:

- **libhpssposix.a** HPSS Posix library

### 1.1.5. Environment Variables

A description of environment variables used by the Client API is provided in this section. In most cases, explicit setting of these environment variables is only required if HPSS was installed with non-default values. Contact your administrator to determine the values being used or refer to the `/var/hpss/etc/env.conf` file for the environment variable settings. The following environment variables can be used to control the Client API's working environment:

The **HPSS\_KRB5\_KEYTAB\_FILE** environment variable defines the name of the file containing the KRB5 security keys necessary for successfully initializing the Client API for servers using Kerberos authentication. The default is `/var/hpss/etc/hpss.keytab`.

The **HPSS\_UNIX\_KEYTAB\_FILE** environment variable defines the name of the file containing the Unix security keys necessary for successfully initializing the Client API for servers using Unix authentication. The default is `/var/hpss/etc/hpss.unix.keytab`.

The **HPSS\_SEC\_REALM\_NAME** environment variable is used to specify the security realm to use when initializing the HPSS security services.

The **HPSS\_API\_DESC\_NAME** environment variable is used to place a descriptive name in any HPSS message logged by the Client API Library. The default value is "Client Application". This variable is only used when logging is enabled in the library.

The **HPSS\_API\_HOSTNAME** environment variable is used to specify the host name to be used for TCP/IP ports created by the Client API. The default value is the default host name of the machine on which the Client API is running. This value can have a significant impact on data transfer performance for data transfers that are handled by the Client API (i.e., those that use the `hpss_Read` and `hpss_Write` interfaces).

The Client API, if compiled with debugging enabled, uses two environment variables to control printing debug information. **HPSS\_API\_DEBUG**, if set to a non-zero value, will enable debug messages. By default, these messages will go to the standard output stream. If **HPSS\_API\_DEBUG\_PATH** is set, however, these messages will be directed to the file indicated by this environment variable. Two special cases for the debug path exist: "stdout" and "stderr", which will use the standard output or standard error I/O streams, respectively.

The **HPSS\_API\_MAX\_CONN** identifies the integer value that will be used for the maximum number of connections allowed. The default is zero, which is equal to the default supported by the HPSS connection management software - currently 150.

The **HPSS\_API\_RETRIES** environment variable is used to control the number of retries to attempt when an operation fails. Currently this class of operation includes library initialization and communications failures. A value of zero indicates that no retries are to be performed, and value of negative one indicates that the operation will be retried until successful. The default value is 4.

The **HPSS\_API\_BUSY\_DELAY** environment variable is used to control the number of seconds to delay between retry attempts. The default value is 15.

The **HPSS\_API\_BUSY\_RETRIES** environment variable is used to control the number of retries to be performed when a request fails because the Core Server does not currently have an available thread to handle that request. A value of zero indicates that no retries are to be performed. A value of negative one indicates that retries should be attempted until either the request succeeds or fails for another reason. The default value is 3.

The **HPSS\_API\_TOTAL\_DELAY** environment variable is used to control the number of total seconds to continue retrying requests. A value of zero indicates that there is no time limit. The default value is 0.

The **HPSS\_API\_LIMITED\_RETRIES** environment variable is used to control the number of retry attempts before a limited retry error operation fails. The default value is 1.

The **HPSS\_API\_RETRY\_STAGE\_INP** environment variable is used to control whether retries are attempted on opens of files in a Class of Service that is configured for background staging on open. A non-zero value indicates that opens which would return -EINPROGRESS to indicate that the file is being staged will be retried (using the same control mechanisms described in the previous paragraph). A value of zero indicates that the -EINPROGRESS return code will be returned to the client. The default value is one (1).

The **HPSS\_API\_REUSE\_CONNECTIONS** environment variable is used to control whether TCP/IP connections are to be left open as long as a file is opened or are to be closed after each read or write request. A non-zero value will cause connections to remain open, while a value of zero will cause connections to be closed. The default value is zero.

The **HPSS\_API\_USE\_PORT\_RANGE** environment variable is used to control whether the HPSS Mover(s) should use the configured port range when making TCP/IP connections for read and write requests. A non-zero value will cause the Mover(s) to use the port range. A value of zero will cause the Mover(s) to allow the operating system to select the port number. The default value is 0.

The **HPSS\_API\_DMAP\_WRITE\_UPDATES** environment variable is used to control the frequency of cache invalidations that are issued to the DMAP file system while writing to a file that is mirrored in HPSS. The default value is 20.

The **HPSS\_API\_DISABLE\_CROSS\_REALM** is used to control cross-realm traversal. When cross-realm traversal is disabled, a client will not be allowed to access directories which are located in another security realm. The default value is 0.

The **HPSS\_API\_DISABLE\_JUNCTIONS** is used to control junction traversal. When junction traversal is disabled, a client will not be allowed to access directories which require traversal to the directory via an HPSS junction. The default value is 0.

The **HPSS\_API\_TRANSFER\_TYPE** environment variable is used to specify the data transport mechanism to be used for data transfers handled by the Client API. The value should be set to "TCP" (for TCP/IP transfers) or "MVRSELECT". The default is MVRSELECT"TCP" (for TCP/IP transfers).

The **HPSS\_API\_SITE\_NAME** environment variable is used to control which HPSS site the client is connected to.

The **HPSS\_API\_AUTHN\_MECH** environment variable is used to control the select of an authentication mechanism.

The **HPSS\_API\_RPC\_PROT\_LEVEL** environment variable is used to control the select of an RPC protection level.

The **HPSS\_API\_SAN3P** environment variable is used to specify whether the SAN3P protocol is being supported.

The **F\_HPSS\_BUFSIZE\_MB** environment variable is used to specify a different buffer size in megabytes for the HPSS buffered I/O functions. The default buffer size is 256MB.

## 1.2. Network Options

The purpose of the Network Options library is to provide an interface to query the information contained in the HPSS network options configuration file for the local machine. This file contains information about network options that may be configured differently for specific network and/or nodes with which the local machine is communicating.

## 1.3. 64-bit Arithmetic Library

### 1.3.1. Purpose

Some HPSS Client APIs require 64-bit fields. The operating system and C compiler on many workstation platforms may not support 64-bit integer operations. As a result, in order to support large integer fields, a set of math libraries have been supplied until 64-bit support is available on all pertinent vendor platforms.

### 1.3.2. Components

The Math Libraries consist of the following macros:

- **add64m**      Add two 64-bit unsigned integers
- **add64\_3m**    Add two 64-bit unsigned integers and store the result in a separate parameter
- **and64m**      Perform a bitwise AND of two 64-bit unsigned integers
- **andbit64m**    AND bit position in 64-bit unsigned integer defined by 32-bit unsigned integer
- **bld64m**      Build a 64-bit unsigned integer from two 32-bit unsigned integers
- **cast64m**      Cast a 32-bit unsigned integer into a 64-bit unsigned integer
- **div64m**      Divide a 64-bit unsigned integer by a 32-bit unsigned integer
- **div2x64m**    Divide a 64-bit unsigned integer by a 64-bit unsigned integer
- **div\_cl64m**    Divide a 64-bit unsigned integer by a 32-bit unsigned integer and return the ceiling
- **div\_2xcl64m** Divide a 64-bit unsigned integer by a 64-bit unsigned integer and return the ceiling
- **eqz64m**      Determine if a 64-bit unsigned integer is zero

- **eq64m** Compare two 64-bit unsigned integers for equality
- **ge64m** Perform greater than or equal to check between two 64-bit unsigned integers
- **gt64m** Perform greater than check between two unsigned 64-bit integers
- **high32m** Extract the high order 32-bits of a 64-bit unsigned integer
- **le64m** Perform less than or equal to check between two 64-bit unsigned integers
- **low32m** Extract the low order 32-bits of a 64-bit unsigned integer
- **lt64m** Perform less than check between two 64-bit unsigned integers
- **mod64m** Modulus a 64-bit unsigned integer by a 32-bit unsigned integer
- **mod2x64m** Modulus a 64-bit unsigned integer by a 64-bit unsigned integer
- **mul64m** Multiply a 64-bit unsigned integer by a 32-bit unsigned integer
- **neqz64m** Determine if a 64-bit unsigned integer is nonzero
- **neq64m** Determine if two 64-bit unsigned integers are not equal
- **not64m** Perform a bitwise NOT of a 64-bit unsigned integer
- **or64m** Perform bitwise OR of two 64-bit unsigned integers
- **shl64m** Shift a 64-bit unsigned integer left by an unsigned 32-bit integer count
- **shl64\_3m** Shift a 64-bit unsigned integer left by an unsigned 32-bit integer count and store in a separate 64-bit unsigned integer
- **shr64m** Shift a 64-bit unsigned integer right by an unsigned 32-bit integer count
- **shr64\_3m** Shift a 64-bit unsigned integer right by an unsigned 32-bit integer count and store in a separate 64-bit unsigned integer
- **sub64m** Subtract a 64-bit unsigned integer from another 64-bit unsigned integer
- **sub64\_3m** Subtract two 64-bit unsigned integers and store the result in a separate parameter.

### 1.3.3. Constraints

The following constraints are being imposed by the 64-bit arithmetic functions:

- 64-bit unsigned integer operations are sufficient, i.e. 64-bit signed arithmetic operations are not supported.
- Multiply functions are limited to 64-bit by 32-bit unsigned operations. For example, a 64-bit unsigned integer may be multiplied by a 32-bit unsigned integer. No 64-bit by 64-bit operations are supported for this category of functions.

### 1.3.4. Libraries

The 64-bit arithmetic functions are included in the **libhpss.a** library.

## 1.4. Storage Concepts

This section defines key HPSS storage concepts which have a significant impact on the usability of HPSS.

Configuration of the HPSS storage objects and policies is the responsibility of your HPSS system administrator.

### 1.4.1. Class of Service

Class of Service (COS) is an abstraction of storage system characteristics that allows HPSS users to select a particular type of service based on performance, space, and functionality requirements. Each COS describes a desired service in terms of characteristics such as minimum and maximum file size, transfer rate, access frequency, latency, and valid read or write operations. A file resides in a particular COS and the class is selected when the file is created. Underlying a COS is a storage hierarchy that describes how data for files in that class are to be stored in HPSS.

COS is specified at file create time. COS hints and priority structures are passed to HPSS in the **hpss\_Open** function. Contact your HPSS system administrator to determine the Classes of Service which have been defined. The following command may also be used to list the defined Classes of Service:

```
lshpss -cos
```

Refer to Chapter 4 of the *HPSS User's Guide* for information on the **lshpss** command. A class of service is implemented by a Storage Hierarchy of one to many Storage Classes. Storage Hierarchies and Storage Classes are not directly visible to the user, but are described below since they map to a Class of Service. The relationship between storage class, storage hierarchy, and COS is shown in *Figure 2: Class of Service / Hierarchy / Storage Class* of the *HPSS Installation Guide*.

### 1.4.2. Storage Class

An HPSS Storage Class is used to group storage media together to provide storage with specific characteristics for HPSS data. The attributes associated with a Storage Class are both physical and logical. Physical media in HPSS are called physical volumes. Physical characteristics associated with physical volumes are the media type, block size, the estimated amount of space on volumes in this class, and how often to write tape marks on the volume (for tape only). Physical media are organized into logical virtual volumes. This allows striping of physical volumes. Some of the logical attributes associated with the Storage Class are virtual volume block size, stripe width, data transfer rate, latency associated with devices supporting the physical media in this class, and storage segment size (disk only). In addition, the Storage Class has attributes that associate it with a particular migration policy and purge policy to help in managing the total space in the Storage Class.

### 1.4.3. Storage Hierarchy

An HPSS Storage Hierarchy consists of multiple levels of storage with each level representing a different storage media (i.e., a storage class). Files are moved up and down the Storage Hierarchy via stage and migrate operations, respectively, based upon storage policy, usage patterns, storage availability, and user request. For example, a Storage Hierarchy might consist of a fast disk, followed by a fast data transfer and medium storage capacity robot tape system, which in turn is followed by a large data storage capacity, but relatively slow data transfer tape robot system. Files are placed on a particular level in the hierarchy depending on the migration policy and staging operations. Multiple copies of a file may also be specified in the migration policy. If data is duplicated for a file at multiple levels in the hierarchy, the more recent data is at the higher level (lowest level number) in the hierarchy. Each hierarchy level is associated with a single storage class.

### 1.4.4. File Family

A file family is an attribute of an HPSS file that is used to group a set of files on a common set of tape virtual volumes. Grouping of files only on tape volumes is supported. Families can only be specified by associating a family with a fileset, and creating the file in the fileset. When a file is migrated from disk to tape, it is migrated to a tape virtual volume assigned to the family associated with the file. If no family is associated with the file, the file is

migrated to the next available tape not associated with a family (actually to a tape associated with family zero). If no tape virtual volume is associated with the family, a blank tape is reassigned from family zero to the file's family. The family affiliation is preserved when tapes are repacked. Configuring file families is an HPSS System Administrator's function.

## 1.5. User Ids

After the HPSS system is configured, the necessary accounts must be created for HPSS users. Contact your HPSS system administrator to add an account. For the **Client API**, either a Unix or Kerberos account must be created. The HPSS system administrator can use the following commands to add a new account.

```
hpssuser -add user -krb
hpssuser -add user -unix
```

## 1.6. Access Control List API

### 1.6.1. Purpose

The access control list API is a set of routines for managing access control lists (ACLs). The routines provide a way to convert ACLs from string format into a form suitable for use by the client API routines. They also provide a way to call the client API routines using ACLs and string format, and a way to convert ACLs back from client API format to string format. In particular, the string conversion routines take care of translating user, group and realm names into UIDs, GIDs and Realm IDs respectively.

### 1.6.2. Components

The access control list library (libhac) contains the following routines:

- **hac\_ConvertACLToHACL** Convert ACL to string format
- **hac\_ConvertHACLToACL** Convert ACL to HPSS format
- **hac\_ConvertHACLToString** Convert ACL to a form suitable for printing
- **hac\_ConvertHACLPermsToPerms** Convert permission string to HPSS format
- **hac\_ConvertHACLTypeToType** Convert ACL entry type to HPSS format
- **hac\_ConvertPermsToHACLPerms** Convert HPSS permission mask to string
- **hac\_ConvertStringsToHACL** Convert ACL strings to HACL format
- **hac\_ConvertTypeToHACLType** Convert ACL entry type to string
- **hac\_DeleteHACL** Delete selected entries from an object's ACL
- **hac\_GetHACL** Get an object's ACL
- **hac\_SetHACL** Replace an object's ACL with a new one
- **hac\_SortHACL** Sort ACL into canonical order
- **hac\_UpdateHACL** Change selected entries in an object's ACL

### 1.6.3. Constraints

None.

### 1.6.4. Libraries

The access control list APIs are available in the HPSS Client Library, **libhpss.a**.

### 1.6.5. Purpose

The Real Time Monitoring API, library routines and utility routines are intended to give system administrators a tool to view the internal activity of the running HPSS system. The RTM library routines enable selected HPSS servers to store information about in-progress requests, while the RTM API in these HPSS servers allows the RTM Utility to collect this information and present it in a readable form.

### 1.6.6. Components

RTM consists of these major parts:

- RTM Library
- RTM API
- RTM Utility

#### The RTM Library

In order to understand how requests are progressing in an HPSS system, a set of library routines (**librtm.a**) are provided for selected HPSS servers to call to store information about each request that the server is handling. The current list of participating servers is the Core Server, Mover, and Gatekeeper. These library routines initialize an in-memory request list **fskip** for each HPSS server, and allow the server to insert, update and delete entries in this list. Associated with each request list entry that the server makes will be a wait list. This wait list will have entries that describe the current wait states of this request. For example, in the Core Server a tape-to-tape copy operation begins as a single request, so a single request entry will be entered into the request list. But ultimately there will be two Core Server threads working on this request. One thread might be waiting on a tape read and the other thread waiting on a tape write. In this case, there would be one request entry for this request id, and two waitlist entries associated with this request entry. On the other hand, within one server there may be more than one request entry for a given request id. This might happen if a tape-to-tape copy enters the Core Server with a read request and a write request. Both requests originate from the Core Server copy request, so both have the same request id. But these requests come to the Core Server separately, so the Core Server would enter a separate request entry onto its request list for each of these requests, and would keep the associated waitlists separate. The **librtm.a** routines are:

- **rtm\_PreInit**
- **rtm\_Init**
- **rtm\_Shutdown**
- **rtm\_ReqListInit**
- **rtm\_ReqListInsertEntry**
- **rtm\_ReqListDeleteEntry**
- **rtm\_ReqListUpdateEntry**

- **rtm\_WaitListInsertEntry**
- **rtm\_WaitListDeleteEntry**
- **rtm\_WaitListUpdateEntry**
- 

## The RTM API

To retrieve the request list information from the selected HPSS servers, each server has an API that can be called by a client external to HPSS. This API interface, **rtm\_GetRequestEntries**, will be exported by each of the selected HPSS servers. Depending upon the parameters to this call, the interface will return varying amounts of information about the requests in a given HPSS server's request list. This interface will be registered using a separate .idl file and assigned its own thread pool so that even if all threads in the server are in use, this request information can be retrieved.

## The RTM Utility

The RTM Utility, **rtmu**, gathers and presents the request information from one or more HPSS servers. Depending upon the nature of the request, the utility calls the **rtm\_GetRequestEntries** API in the appropriate HPSS server(s) and presents the returned request data in a short, long or script-ready output format. For example, if all requests active in the Core Server are requested, this utility binds to the Core Server and calls **rtm\_GetRequestEntries** to retrieve the desired information. Or if the utility is told to retrieve all information in the HPSS system about a given request, the utility starts by binding to the Core Server and calling **rtm\_GetRequestEntries**; and depending upon the results of this request, the utility may contact any associated movers. There is a variety of request options.

### 1.6.7. Constraints

The following constraints are being imposed upon HPSS as a result of this subsystem design:

- The HPSS MVR will not be able to use all of this design approach because of using separate processes instead of threads, and because of the requirement for a non-DCE interface for the mover.

### 1.6.8. Libraries

Applications calling the RTM library routines must link with the following library:

- librtm.a

Applications calling the RTM API **rtm\_GetRequestEntries** must link with the following libraries:

- libgss.a
- libhandles.a
- libhpssapi.a
- libhpssxdr.a
- libhpsscomm.a
- libhpsscs.a
- libhpsslog.a
- libhpsslsl.a



- libhpssavlib.a
- libhsec.a
- libinterop.a
- libmetadata.a
- libpdata.a
- librtm.a
- libtraniod.a



## Chapter 2. *Client API Functions*

This chapter specifies the HPSS client programming interface. Specifically, the following information is provided:

- Standard Application Programming Interfaces (APIs) - **API Interfaces**
- Structure definitions - **Data Definitions**

### 2.1. API Interfaces

This section describes all API interfaces which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

- Name
- Synopsis
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Notes

Note that for each thread that issues an HPSS Client API call, a call must be made to **hpss\_ThreadCleanUp** with the thread id for that thread. This is necessary so that the client API can free state and memory allocated to that thread.

Note that there are a number of errors that may be returned from a Client API call which are not actually errors generated by performing the call, but are caused by a failure of the client API to successfully initialize. These values may be returned from any routine and include:

EAGAIN	An HPSS server is not ready or received a communication error, and the request could be retried.
ENOCONNECT	The Client API could not connect to either the Location Server or the Core Server.
ENOMEM	Memory could not be allocated for internal Client API State.
EPERM	The user's client credentials could not be established.
EIO	An internal HPSS error occurred.
ESTALE	The open file or directory is no longer valid - close and reopen the file or directory to reestablish a valid open descriptor. This error is likely due to the connections to the HPSS servers being reset.
ETIMEDOUT	An HPSS server request timed out or received a communication error, and the request could not be successfully retried.

#### 2.1.1. **hpss\_Access**

##### **Purpose**

Check file accessibility.

## Synopsis

```
#include <unistd.h>
#include "hpss_api.h"

int
hpss_Access(
    char *Path,      /* IN */
    int  Amode );    /* IN */
```

## Description

The **hpss\_Access** function checks the accessibility of the file named by *Path* for the file access indicated by *Amode*. Refer to POSIX.1 for more detailed information.

## Parameters

<i>Path</i>	Points to the path name of the file for which client accessibility is being checked.
<i>Amode</i>	Indicates the type of file access being checked. Refer to POSIX.1 for possible values.

## Return Values

If the requested access is permitted, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an *errno* value set by POSIX.1 access.

## Error Conditions

EACCES	The permissions specified by <i>Amode</i> are denied, or search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	An invalid value was specified for <i>Amode</i> .
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_AccessHandle, hpss\_Chmod, hpss\_FileSetAttributes.**

## Notes

None.

## 2.1.2. hpss\_AccessHandle

### Purpose

Check file accessibility.

## Synopsis

```
#include <unistd.h>
#include "hpss_api.h"
int
hpss_AccessHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char              *Path,         /* IN */
    int               *Amode,         /* IN */
    sec_cred_t        *Ucred,        /* IN */
    hpss_authz_token_t *AuthzTicket); /* OUT */
```

## Description

The **hpss\_AccessHandle** function checks the accessibility of the file named by *Path* for the file access indicated by *Amode*. Refer to POSIX.1 for more detailed information.

## Parameters

<i>ns_ObjHandle_t</i>	Pointer to handle of the parent object
<i>Path</i>	Points to the path name of the file for which client accessibility is being checked.
<i>Amode</i>	Indicates the type of file access being checked. Refer to POSIX.1 for possible values.
<i>Ucred</i>	Pointer to User Credentials.
<i>AuthzTicket</i>	Returned pointer to authorization ticket.

## Return Values

If the requested access is permitted, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an *errno* value set by POSIX.1 access.

## Error Conditions

EACCES	The permissions specified by <i>Amode</i> are denied, or search permission is denied on a component of the path prefix.
EINVAL	An invalid value was specified for <i>Amode</i> or the <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Access**, **hpss\_Chmod**, **hpss\_FileSetAttributes**.

## Notes

None.

## 2.1.3. hpss\_AcctCodeToName

### Purpose

Finds the Account Name associated with the given Account Code

### Synopsis

```
#include "hpss_api.h"
int
hpss_AcctCodeToName(
    acct_rec_t    AcctCode,    /* IN      */
    hpss_uuid_t   *Site,       /* IN/OUT  */
    char          *AcctName);  /* OUT     */
```

### Description

Finds and returns the Account Name associated with a given Account Code.

### Parameters

<i>AcctCode</i>	The Account Code to look up.
<i>Site</i>	Pointer to an area that contains the UUID of the site you are interested in. If this UUID is zeroed out or null, the index for the site managing the current working directory is used.
<i>AcctName</i>	The Account Name associated with the given Account Code. Account Name should be a string of at least HPSS_MAX_ACCOUNT_NAME (128) characters.

### Return Values

Upon successful completion, hpss\_AcctCodeToName returns zero. Otherwise, hpss\_AcctCodeToName returns a negative value, the absolute value of which indicates the specific error.

### Error Conditions

EFAULT	The AcctName is a NULL pointer or UUID compare for site failed.
EINVAL	The given AccountCode is not valid.

### See Also

**hpss\_AcctNameToCode**

### Notes

None.

## 2.1.4. hpss\_AcctNameToCode

### Purpose

Finds the Account Code associated with the given Account Name

### Synopsis

```
#include "hpss_api.h"
```

```

int
hpss_AcctNameToCode(
    char      *AcctName,      /* IN/OUT */
    hpss_uuid_t *Site,        /* IN/OUT */
    acct_rec_t *AcctCode );   /* OUT    */

```

### Description

Finds and returns the Account Code associated with a give Account Name.

### Parameters

<i>AcctName</i>	Pointer to the Account Name to look up.
<i>Site</i>	Pointer to an area that contains the UUID of the site. If this UUID is zeroed out or null, the name of the site managing the current working directory is used.
<i>AcctCode</i>	Pointer to the Account Code associated with the given Account Name.

### Return Values

Upon successful completion, hpss\_AcctNameToCode returns zero. Otherwise, hpss\_AcctNameToCode returns a negative value, the absolute value of which indicates the specific error.

### Error Conditions

EINVAL	The given Account Name is not valid.
EFAULT	The account name or account code is a NULL pointer.
ENAMETOOLONG	The account name is too long.

### See Also

**hpss\_AcctCodeToName**

### Notes

None.

## 2.1.5. hpss\_BeginExTrans

### Purpose

Begin an HPSS extended transaction.

### Synopsis

```

#include "hpss_api.h"
int
hpss_BeginExTrans(
    hpss_uuid_t      *CoreServerUUID); /* IN */

```

### Description

This routine will begin an HPSS extended transaction for the specified Core Server. All transaction enabled APIs will be made within the current transaction for the thread. A transaction begun by this

routine will remain current for the thread until an `hpss_EndExTrans()` call is issued from the same thread.

#### Parameters

*CoreServerUUID*                      Core Server UUID.

#### Return Values

Upon successful completion, `hpss_BeginExTrans` returns zero. Otherwise, `hpss_BeginExTrans` returns a negative value, the absolute value of which indicates the specific error.

#### Error Conditions

`EINVAL`                              `CoreServerUUID` NULL.  
`ENOTDIR`                             A non-DMGateway client issued the API.

#### See Also

**`hpss_EndExTrans`**

#### Notes

None.

## 2.1.6. `hpss_Chacct`

#### Purpose

Change the account code of an HPSS file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Chacct(
    char          *Path,          /* IN */
    acct_rec_t    AcctCode);     /* IN */
```

#### Description

The `hpss_Chacct` routine changes the accounting code for the file or directory named by *Path*.

#### Parameters

*Path*                                Names the file for which the account code is being changed.  
*AcctCode*                            Specifies the new accounting code for the file.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

#### Error Conditions

`EACCES`                              Search permission is denied on a component of the path prefix.  
`EFAULT`                             The *Path* parameter is a NULL pointer.



ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation or is configured for Unix-style accounting.

#### See Also

**hpss\_AcctCodeToName, hpss\_AcctNameToCode, hpss\_ChacctByName, hpss\_GetAcct, hpss\_GetAcctName, hpss\_SetAcct, hpss\_SetAcctByName, hpss\_Chown, hpss\_SetFileAttributes**

#### Notes

None.

## 2.1.7. hpss\_ChacctByName

#### Purpose

Change the account code of an HPSS file or directory by specifying the account's name.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_ChacctByName(
    char    *Path,           /* IN */
    char    *AccountName ); /* IN */
```

#### Description

The hpss\_ChacctByName routine changes the accounting code for the file or directory named by *Path*.

#### Parameters

<i>Path</i>	Names the file or directory for which the account code is being changed.
<i>AccountName</i>	The account name corresponding to the new accounting code for the file or directory.

#### Return Values

Upon successful completion, a value of zero is returned . Otherwise, a negative value is returned, the absolute value of which indicated the specific error.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer or AcctName is NULL.
EINVAL	The specified <i>AccountName</i> is not a valid account name.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have appropriate privilege to perform the operation or is configured for Unix-style accounting.

#### See Also

**hpss\_GetAcct, hpss\_SetAcct, hpss\_Chown, hpss\_SetFileAttributes, hpss\_GetAcctName, hpss\_SetAcctByName, hpss\_Chacct**

#### Notes

None.

## 2.1.8. hpss\_Chdir

#### Purpose

Change current working directory.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Chdir(
    char    *Path );          /* IN */
```

#### Description

The **hpss\_Chdir** function changes a thread's current working directory to be the directory named by *Path*.

#### Parameters

<i>Path</i>	Specifies path name of the directory to which the current working directory is to be changed.
-------------	---

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **chdir**.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_Getcwd, hpss\_Chroot.**

## Notes

None.

## 2.1.9. hpss\_Chmod

### Purpose

Change the file mode of an HPSS file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Chmod(
    char    *Path, /* IN */
    mode_t  Mode); /* IN */
```

### Description

The **hpss\_Chmod** function alters the file mode associated with file named by *Path*.

### Parameters

<i>Path</i>	Points to the path name of the file for which the file mode is being changed.
<i>Mode</i>	Specifies the new value to which the file mode is to be set.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **chmod**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

### See Also

**hpss\_Chown, hpss\_Stat, hpss\_FileGetAttributes, hpss\_FileSetAttributes.**

## Notes

None.

## 2.1.10. hpss\_Chown

### Purpose

Change owner and group of an HPSS File.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Chown(
    char    *Path,          /* IN */
    uid_t   Owner,          /* IN */
    gid_t   Group );       /* IN */
```

### Description

The **hpss\_Chown** function sets the user ID and group ID of the file named by *Path* to the values specified by *Owner* and *Group*, respectively.

### Parameters

<i>Path</i>	Names the file for which the owner and group owner are being changed.
<i>Owner</i>	Specifies the new value for the owner of the file.
<i>Group</i>	Specifies the new value for the group owner of the file.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **chown**. If the owner is changed, the account code of the file or directory will also be changed to reflect that configured for the new owner.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

### See Also

**hpss\_Chmod**, **hpss\_Stat**, **hpss\_FileGetAttributes**, **hpss\_FileSetAttributes**.

### Notes

None.

## 2.1.11. hpss\_Chroot

### Purpose

Change the root directory for the current client.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Chroot(
    char *Path );    /* IN */
```

## Description

The **hpss\_Chroot** function changes the root directory for the current client. After a successful call to **hpss\_Chroot**, all absolute path name operations are done relative to *Path*, and relative operations cannot be made out of the subtree whose root is *Path*.

## Parameters

<i>Path</i>	Specifies the path name of the directory that is to become the new root directory.
-------------	--

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	This call was made from the nonglobal Client API library.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Chdir**, **hpss\_Getcwd**.

## Notes

Note that as currently implemented, symbolic links could allow a client to access files outside the new root directory (since **hpss\_Chroot** bookkeeping is maintained entirely in the client API but symbolic links are generally handled in the Core Server). If this is a problem (the only current projected client is the FTP server for anonymous FTP), changes could be made to ensure that the symbolic links do not access files outside the subtree - failing if they do, or possibly traversing the symbolic link contents within the client API.

## 2.1.12. hpss\_ClientAPIInit

### Purpose

Initialize the Client API for the current thread. This is usually done automatically during the first Client API call made. However, it may also be performed separately, before other Client API calls.

### Synopsis

```
#include "hpss_api.h"
```

```

int
hpss_ClientAPIInit(
    apithrdstate_t      **ThreadContext );   /* OUT */

```

### Description

If the thread's state has already been initialized just return a pointer to the thread's specific state. Otherwise initialize and connect to all necessary servers and then return the new thread specific state.

### Parameters

<i>ThreadContext</i>	Thread's specific state
----------------------	-------------------------

### Return Values

0	No error. Thread specific state is valid.
---	---

### Error Conditions

ENOCONNECT	Could not initialize HPSS connection services.
ENOMEM	Memory allocation failed.
EPERM	Could not initialize security.

### See Also

None.

### Notes

None.

## 2.1.13. hpss\_ClientAPIReset

### Purpose

Reset the current Client API control information.

### Synopsis

```

#include "hpss_api.h"
void
hpss_ClientAPIReset(void);

```

### Description

The **hpss\_ClientAPIReset** routine will clean up the current Client API control information, including closing server connections. The next Client API call should then reinitialize the control information based on the current configuration information.

### Parameters

None.

### Return Values

None.

## Error Conditions

None.

## See Also

**hpss\_GetConfiguration**, **hpss\_SetConfiguration**.

## Notes

None.

## 2.1.14. hpss\_Close

### Purpose

Close a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Close(
    int    Fildes );    /* IN */
```

### Description

The **hpss\_Close** function terminates the connection between the file handle, *Fildes*, and the file to which it is associated. The file handle and any associated resources are deallocated and can be reused by a subsequent call to **hpss\_Open**.

### Parameters

*Fildes*                      Specifies the file handle obtained from a previous **hpss\_Open**.

### Return Values

Upon successful completion, **hpss\_Close** returns zero. Otherwise, **hpss\_Close** returns a negative value; the absolute value of which is equal to an errno value set by POSIX.1 **close**.

### Error Conditions

EBADF                      The specified file descriptor is out of range, or does not refer to an open file.  
EBUSY                      The file is currently in use by another client thread.

### See Also

**hpss\_Open**, **hpss\_OpenBitfile**, **hpss\_ReopenBitfile**.

### Notes

None.

## 2.1.15. hpss\_Closedir

### Purpose

Close an open directory stream.

## Synopsis

```
#include "hpss_api.h"
int
hpss_Closedir(
    int    Dirdes );    /* IN */
```

## Description

The **hpss\_Closedir** function closes the directory stream corresponding to the open directory stream handle *Dirdes*.

## Parameters

*Dirdes*                      Specifies the open directory stream handle corresponding to the stream to be closed.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 *closedir*.

## Error Conditions

EBADF                      The specified directory descriptor does not refer to an open directory.  
EBUSY                      The directory is currently in use by another client thread.

## See Also

**hpss\_Opendir**.

## Notes

None.

## 2.1.16. hpss\_ConvertIdsToNames

### Purpose

Convert UIDs to user names, GIDs to group names, and/or realm IDs to realm names.

### Synopsis

```
int
hpss_ConvertIdsToNames(
    signed32      NumEntries, /* IN */
    api_namespec_t *Specs ); /* IN/OUT */
```

### Description

The *hpss\_ConvertIdsToNames* function converts a batch of Unix UIDs to user names, GIDs to group names, and/or realm ids to realm names. Each entry in the *Specs* array tells what kind of translation is needed for that entry.

### Parameters

*NumEntries*                      Number of entries in the *Specs* array.



<i>Specs</i>	Pointer to an area that contains information for defining a principal, which is to be converted, and the results of the conversion.
--------------	---

## Return Values

Upon successful completion this routine returns zero. Otherwise it returns an error value describing the problem.

## Error Conditions

EINVAL	The <i>Specs</i> array has an entry that contains a zero realm id.
ENOCONNECT	A problem with the security registry.
EFAULT	A name is too long to fit into <i>Specs</i> .
EAGAIN	Default for security registry error.

## See Also

### hpss\_ConvertNamesToIds

## Notes

1. The *Specs* array must contain *NumEntries* elements of type *api\_name\_spec\_t*. Each array element has a *Type* field that determines how the element will be translated. If the *Type* is NAMESPEC\_SKIP, then the *Id* and *RealmId* fields will be ignored, and the *Name* and *RealmName* fields will be set to undefined values. NAMESPEC\_SKIP enables ACL editors to deal with ACL entry types, such as the mask object, which do not contain any ids.
2. If the *Type* is set to NAMESPEC\_REALM, then the *RealmId* field will be translated into a realm name, which is returned in the *RealmName* field. In this case, the *Id* field will be ignored and the *Name* field will be set to an undefined value. On the other hand, if the *Type* is set to NAMESPEC\_USER or NAMESPEC\_GROUP, then the function returns the user or group name in the *Name* field as well as filling in the *RealmName* field.
3. If, at any point during the translation process, an array entry is found that cannot be translated, the routine will return immediately. In this case, the *Name* and *RealmName* fields of each *Specs* array entry should be considered undefined, but the *Type*, *Id* and *RealmId* fields will remain unchanged from their initial values. Therefore the caller may need to keep a copy of the *Specs* array if the name fields will be needed again.
4. If the principal cannot be found in the desired realm, then the routine does not return an error, but rather just stores an ASCII representation of the principal's id in the *Name* field. Likewise, if the realm cannot be found in the trusted realm table, then the routine returns the realm id in the *RealmName* field. This allows the caller to deal with principals and/or realms that no longer exist. The caller may detect this "error" by scanning for numeric data in these fields.

## 2.1.17. hpss\_ConvertNamesToIds

### Purpose

Convert user names to UIDs, group names to GIDs , and/or realm names to realm IDs.

### Synopsis

```
int
hpss_ConvertNamesToIds (
```

```
signed32      NumEntries, /* IN */
api_namespec_t *Specs ); /* IN/OUT */
```

## Description

The *hpss\_ConvertNamesToIds* function converts an array of user names to UIDs, group names to GIDs, and/or realm names to realm IDs. Each entry in the *Specs* array tells what kind of translation is needed for that entry.

## Parameters

<i>NumEntries</i>	Number of entries in the <i>Specs</i> array.
<i>Specs</i>	Pointer to an area that contains information for defining a principal, which is to be converted, and the results of the conversion.

## Return Values

Upon successful completion this routine returns zero. Otherwise it returns an error value describing the problem.

## Error Conditions

ENOENT	One or more entries in the <i>Specs</i> array specified a principal or realm name that could not be found.
ENOCONNECT	A problem with the security registry.
EINVAL	Non-numeric principal id was specified.

## See Also

**hpss\_ConvertIdsToNames**

## Notes

1. The *Specs* array must contain *NumEntries* elements of type *api\_name\_spec\_t*. Each array entry has a *Type* field that determines how that element will be translated. If the element's *Type* is NAMESPEC\_SKIP, then the *Name* and *RealmName* fields will be ignored and the *Id* and *RealmId* fields will be set to undefined values. This is used to help ACL editors deal with ACL entry types, such as the mask object, which do not contain any names.
2. If the element's *Type* is set to NAMESPEC\_REALM, the *RealmName* field is translated into a realm id, which is returned in the *RealmId* field. In this case, the *Name* field will be ignored and the *Id* field will be set to an undefined value. On the other hand, if *Type* is set to NAMESPEC\_USER or NAMESPEC\_GROUP, the function returns the UID or GID in the *Id* field as well as filling in the *RealmId* field. If the element's *Name* and/or *RealmName* fields are strings or digits, the routine does not try to verify that the principal and/or realm actually exist. Rather, it just returns the corresponding values in the *Id* and *RealmId* fields. This allows the caller to deal with principals and/or realms that no longer exist. In this situation, the routine does not return ENOENT.
3. If at any point during the translation process, an array entry is found that cannot be translated, the routine will return immediately. In this case, the *Id* and *RealmId* fields of each *Specs* array entry should be considered undefined, but the *Type*, *Name*, and *RealmName* fields will remain unchanged from their initial values. Therefore the caller may need to keep a copy of the *Specs* array if the id fields will be needed again.

## 2.1.18. hpss\_CopyFile

### Purpose

Copy an HPSS file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_CopyFile(
    int    SrcFiledes,      /* IN */
    int    DestFiledes)    /* IN */
```

### Description

The **hpss\_CopyFile** function copies source file to destination file.

### Parameters

<i>SrcFiledes</i>	File descriptor of open source file.
<i>DestFiledes</i>	File descriptor of open destination file.

### Return Values

Upon successful completion, **hpss\_CopyFile** returns zero. Otherwise a negative value is returned; the absolute value of which is equal to an errno value defined below.

### Error Conditions

EBADF	Source or destination file descriptors out of range (i.e. negative) or not HPSS type file descriptors, or the destination file was not opened for writes.
EBUSY	Another thread is currently manipulating the destination file.
EIO	An internal HPSS error has occurred.
ENOTSUP	Source and destination files not owned by same Core Server.
ESTALE	Connection for destination file no longer valid.

### See Also

**hpss\_Open**

### Notes

None.

## 2.1.19. hpss\_Creat

### Purpose

Create an HPSS file and open it for writing.

### Synopsis

```
#include "hpss_api.h"
int
```

```

hpss_Creat(
    char                *Path,                /* IN */
    mode_t              Mode,                /* IN */
    hpss_cos_hints_t    *HintsIn,            /* IN */
    hpss_cos_priorities_t *HintsPri,         /* IN */
    hpss_cos_hints_t    *HintsOut );         /* OUT */

```

## Description

The **hpss\_Creat** function calls **hpss\_Open** with the *Oflag* parameter defined as *O\_WRONLY | O\_CREAT | O\_TRUNC*. It creates a new file or rewrites an existing one. The newly created or previously existing file is opened for writing (see **hpss\_Open**).

## Parameters

<i>Path</i>	Names the file to be opened or created.
<i>Mode</i>	Specifies the file permission.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsIn</i> structure. This argument may be a NULL pointer and the priority will be set to <i>REQUIRED_PRIORITY</i> .
<i>HintsOut</i>	Points to an <b>hpss_cos_hints_t</b> structure which will contain the values actually used when the file is created. This argument may be a NULL pointer if the returned values are to be ignored.

## Return Values

Upon successful completion, **hpss\_Creat** returns a file descriptor. Otherwise, it returns a negative value; the absolute value of which is equal to an *errno* value (see **hpss\_Open**).

## Error Conditions

See **hpss\_Open** for error conditions.

## See Also

**hpss\_Open**, **hpss\_Umask**.

## Notes

The **hpss\_Creat** function mimics the POSIX **creat** function. It opens the file for writing and returns a file descriptor.

## 2.1.20. hpss\_Create

### Purpose

Create an HPSS file.

### Synopsis

```
#include "hpss_api.h"
```

```

int
hpss_Create (
    char                *Path,                /* IN */
    mode_t              Mode,                /* IN */
    hpss_cos_hints_t    *HintsIn,            /* IN */
    hpss_cos_priorities_t *HintsPri,         /* IN */
    hpss_cos_hints_t    *HintsOut );         /* OUT */

```

## Description

The **hpss\_Create** function creates the specified file, if it does not already exist. The newly created or previously existing file is not opened (see **hpss\_Open**).

## Parameters

<i>Path</i>	Names the file to be opened or created.
<i>Mode</i>	Specifies the file permission.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsIn</i> structure. This argument may be a NULL pointer and the priority will be set to <b>REQUIRED_PRIORITY</b> .
<i>HintsOut</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsIn</i> structure. This argument may be a NULL pointer and the priority will be set to <b>REQUIRED_PRIORITY</b> .

## Return Values

Upon successful completion, **hpss\_Create** returns zero. Otherwise, **hpss\_Create** returns a negative value; the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the <i>Path</i> prefix or the file does not exist and write permission is denied for the parent directory of the file to be created.
EAGAIN	Gatekeeper retries have timed out.
EEXIST	The named file exists.
EFAULT	Path is NULL.
EINVAL	One or more values in the <i>HintsIn</i> parameter is invalid.
ENAMETOOLONG	The length of the <i>Path</i> string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	Path is an empty string.
ENOMEM	Memory could not be allocated for path name.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Open**, **hpss\_Umask**.

## Notes

The **hpss\_Create** function differs from the POSIX **creat** in that no attempt is made to open the file and it behaves as if the **O\_EXCL** flag was set (see **EEXIST**, above).

## 2.1.21. hpss\_CreateDMHandle

### Purpose

Create an HPSS DMAP file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_CreateDMHandle(
    ns_ObjHandle_t      ObjHandle,          /* IN */
    char                *Path,              /* IN */
    mode_t              Mode,                /* IN */
    sec_cred_t          *Ucred,             /* IN */
    hpss_cos_hints_t     *HintsIn,           /* IN */
    hpss_cos_priorities_t *HintsPri,         /* IN */
    unsigned32          ConsistencyFlags,    /* IN */
    hpss_cos_hints_t     *HintsOut,          /* OUT */
    hpss_vattr_t         *AttrsOut,          /* OUT */
    hpss_authz_token_t   *AuthzTicket)       /* OUT */
```

### Description

The **hpss\_CreateDMHandle** function creates a DMAP file specified by 'Path', with permissions as specified by 'Mode' and using the class of service values specified by 'HintsIn' and 'HintsPri', if non-NULL.

### Parameters

<i>ObjHandle</i>	NS object handle of parent directory of Path.
<i>Path</i>	Names the file to be opened or created.
<i>Mode</i>	Specifies the file mode used for determining the mode for the created file.
<i>Ucred</i>	Pointer to structure containing the user credentials for the new file.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsIn</i> structure. This argument may be a NULL pointer and the priority will be set to <b>REQUIRED_PRIORITY</b> .
<i>ConsistencyFlags</i>	Core Server consistency flags to use for the new file.
<i>HintsOut</i>	Points to an <b>hpss_cos_hints_t</b> structure which will contain the values actually

used when the file is created. This argument may be a NULL pointer if the returned values are to be ignored.

*AttrsOut*

Pointer to structure containing the attributes for the new file.

*AuthzTicket*

Pointer to structure containing a client authorization ticket for the new file.

## Return Values

Upon successful completion, **hpss\_CreateDMHandle** returns zero. Otherwise, a negative value is returned; the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the <i>Path</i> prefix or the file does not exist and write permission is denied for the parent directory of the file to be created.
EAGAIN	Gatekeeper retries have timed out.
EEXIST	The named file exists.
EFAULT	Path is NULL.
EINVAL	One or more values in the <i>HintsIn</i> parameter is invalid or the ObjHandle is
ENAMETOOLONG	The length of the <i>Path</i> string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	Path points to a zero value.
ENOSPC	Resources could not be allocated for the new file.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Create**

## Notes

None.

## 2.1.22. hpss\_CreateHandle

### Purpose

Create an HPSS file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_CreateHandle(
    ns_ObjHandle_t    ObjHandle,    /* IN */
    char               *Path,        /* IN */
    mode_t             Mode,         /* IN */
    sec_cred_t         *Ucred,       /* IN */
    hpss_cos_hints_t   *HintsIn,     /* IN */
    hpss_cos_priorities_t *HintsPri, /* IN */
    ...)
```

```

hpss_cos_hints_t          *HintsOut,          /* OUT */
hpss_vattr_t             *AttrsOut,         /* OUT */
hpss_authz_token_t       *AuthzTicket)      /* OUT */

```

## Description

The **hpss\_CreateHandle** function creates a file specified by 'Path', with permissions as specified by 'Mode' and using the class of service values specified by 'HintsIn' and 'HintsPri', if non-NULL. The newly created or previously existing file is not opened (see **hpss\_Open**).

## Parameters

<i>ObjHandle</i>	NS object handle of parent directory of Path.
<i>Path</i>	Names the file to be created.
<i>Mode</i>	Specifies the file mode used for determining the mode for the created file.
<i>Ucred</i>	Pointer to an <b>sec_cred_t</b> structure containing the user credentials for the new file.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsIn</i> structure. This argument may be a NULL pointer and the priority will set to REQUIRED_PRIORITY.
<i>HintsOut</i>	Points to an <b>hpss_cos_hints_t</b> structure which will contain the values actually used when the file is created. This argument may be a NULL pointer if the returned values are to be ignored.
<i>AttrsOut</i>	Pointer to an <b>hpss_vattr_t</b> structure containing the attributes for the new file.
<i>AuthzTicket</i>	Pointer to an <b>hpss_authz_token_t</b> structure containing a client authorization ticket for the new file.

## Return Values

Upon successful completion, **hpss\_CreateHandle** returns zero. Otherwise, a negative value is returned; the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the <i>Path</i> prefix or the file does not exist and write permission is denied for the parent directory of the file to be created.
EAGAIN	Gatekeeper retries have timed out.
EEXIST	The named file exists.
EFAULT	Path is NULL.
EINVAL	One or more values in the <i>HintsIn</i> parameter is invalid or the <i>ObjHandle</i> is NULL.
ENAMETOOLONG	The length of the <i>Path</i> string exceeds the system-imposed path name limit or a



	path name component exceeds the system-imposed limit.
ENOENT	Path points to a zero value.
ENOSPC	Resources could not be allocated for the new file.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_Create**

#### Notes

None.

## 2.1.23. hpss\_CreateWithHints

#### Purpose

Create an HPSS file with specified COS and priority.

#### Synopsis

```
#include "hpss_posix_api.h"
int
hpss_CreateWithHints(
    char                *Path,           /* IN */
    mode_t              Mode,           /* IN */
    hpss_cos_hints_t    *HintsIn,       /* IN */
    hpss_cos_priorities_t *HintsPri,    /* IN */
    hpss_cos_hints_t    *HintsOut);     /* OUT */
```

#### Description

The **hpss\_CreateWithHints** function creates a file specified by *Path*, with permissions as specified by *Mode* and using the class of service values specified by *HintsIn* and *HintsPri*, if non-NULL. The newly created or previously existing file is not opened (see **hpss\_Open**).

#### Parameters

<i>Path</i>	Names the file to be created.
<i>Mode</i>	Specifies the file mode used for determining the mode for the created file.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsIn</i> structure. This argument may be a NULL pointer and the priority will set to REQUIRED_PRIORITY.
<i>HintsOut</i>	Points to an <b>hpss_cos_hints_t</b> structure which will contain the values actually used when the file is created. This argument may be a NULL pointer if the returned values are to be ignored.

#### Return Values

Upon successful completion, **hpss\_CreateWithHints** returns zero. Otherwise, a negative value is returned; the absolute value of which is equal to an errno value, as defined below.

#### Error Conditions

EACCES	Search permission is denied on a component of the <i>Path</i> prefix or the file does not exist and write permission is denied for the parent directory of the file to be created.
EAGAIN	Gatekeeper retries have timed out.
EEXIST	The named file exists.
EFAULT	Path is NULL.
EINVAL	One or more values in the <i>HintsIn</i> parameter is invalid.
ENAMETOOLONG	The length of the <i>Path</i> string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	Path points to an empty string.
ENOMEM	Memory could not be allocated for the new psth name.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_Create**

#### Notes

The **hpss\_CreateWithHints** function is identical to **hpss\_Create**. It is created at the request of a customer.

## 2.1.24. hpss\_DeleteACL

#### Purpose

Removes entries from the Access Control List of a file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_DeleteACL(
    char            *Path,          /* IN */
    unsigned32      Options,        /* IN */
    ns_ACLConfArray_t *ACL );      /* IN */
```

#### Description

The **hpss\_DeleteACL** function removes the ACL entries specified by *ACL* from the file named by *Path*.

#### Parameters

<i>Path</i>	Names the file for which the <i>ACL</i> is being removed.
<i>Options</i>	Bit vector used to specify what type of ACL is to be retrieved. One of: <ul style="list-style-type: none"><li>● HPSS_ACL_OPTION_OBJ – return object's normal ACL.</li></ul>

- HPSS\_ACL\_OPTION\_IO – return the initial-object ACL. (only valid for directory objects).
- HPSS\_ACL\_OPTION\_IC – return the initial-container ACL. (only valid for directory objects).

*ACL*

Points to the list of ACL entries to be removed.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned; the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	Exactly one of the HPSS_ACL_OPTION_* bits must be set in the <i>Options</i> bit vector to avoid receiving this error.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.
ESRCH	A specified ACL entry did not match an existing ACL entry for the file.

## See Also

**hpss\_GetACL, hpss\_SetACL, hpss\_UpdateACL.**

## Notes

None.

## 2.1.25. hpss\_DeleteACLHandle

### Purpose

Removes entries from the Access Control List of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_DeleteACLHandle(
    ns_ObjHandle_t    ObjHandle,    /* IN */
    char               *Path,        /* IN */
    sec_cred_t         *Ucred,       /* IN */
    unsigned32         Options,      /* IN */
    ns_ACLConfArray_t *ACL );        /* IN */
```

### Description

The **hpss\_DeleteACLHandle** function removes an ACL entry from the Access Control List of a file or directory named by 'Path', taken relative to the directory indicated by 'ObjHandle'.

#### Parameters

<i>ObjHandle</i>	Parent object handle.
<i>Path</i>	Names the file for which the <i>ACL</i> is being removed.
<i>Ucred</i>	User credentials.
<i>Options</i>	Bit vector used to specify what type of ACL is to be retrieved. One of: <ul style="list-style-type: none"><li>● HPSS_ACL_OPTION_OBJ – return object's normal ACL.</li><li>● HPSS_ACL_OPTION_IO – return the initial-object ACL. (only valid for directory objects).</li><li>● HPSS_ACL_OPTION_IC – return the initial-container ACL. (only valid for directory objects).</li></ul>
<i>ACL</i>	Points to the list of ACL entries to be removed.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned; the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	Exactly one of the HPSS_ACL_OPTION_* bits must be set in the <i>Options</i> bit vector to avoid receiving this error.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.
ESRCH	A specified ACL entry did not match an existing ACL entry for the file.

#### See Also

**hpss\_GetACL**, **hpss\_SetACL**, **hpss\_UpdateACL**.

#### Notes

None.

## 2.1.26. hpss\_EndExTrans

#### Purpose

End an HPSS extended transaction.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_EndExTrans(
    hpss_ExTransOutcome_t    Outcome)    /* IN */
```

### Description

This routine will end the current HPSS extended transaction for the thread.

### Parameters

*Outcome*                      Desired transaction outcome (Abort or Commit).

### Return Values

Upon successful completion, `hpss_EndExTrans` returns zero. Otherwise, `hpss_EndExTrans` returns a negative value, the absolute value of which indicates the specific error.

### Error Conditions

EINVAL	Invalid <i>Outcome</i> specified.
ENOENT	No current transaction for the thread.
ENOTSUP	A non-DMAP Gateway client issued the API.

### See Also

**hpss\_BeginExTrans**

### Notes

None.

## 2.1.27. hpss\_Fclear

### Purpose

Clear part of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Fclear(
    int            Fildes,    /* IN */
    u_signed64    Length );    /* IN */
```

### Description

The **hpss\_Fclear** routine clears part of an open file, specified by *Fildes*, the current file offset and *Length*. A hole will be created in the file covering the part of the file that was cleared, and its storage resource may be freed accordingly.

### Parameters

*Fildes*                      Specifies the file descriptor identifying the open file for which part is to be cleared.

*Length* Specifies the number of bytes to be cleared.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an `errno` value defined below.

#### Error Conditions

<code>EBADF</code>	The specified file descriptor does not correspond to a file opened for writing.
<code>EBUSY</code>	The specified file descriptor is currently in use.

#### See Also

`hpss_FclearOffset`, `hpss_Truncate`, `hpss_Ftruncate`.

#### Notes

None.

## 2.1.28. `hpss_FclearOffset`

#### Purpose

Clear part of a file beginning at the specified offset.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_FclearOffset(
    int      Fildes,      /* IN */
    u_signed64 Offset,    /* IN */
    u_signed64 Length ); /* IN */
```

#### Description

The `hpss_FclearOffset` routine clears part of an open file, specified by *Fildes*, the current file *Offset*, and *Length*. A hole will be created in the file covering the part of the file that was cleared, and storage resources may be freed accordingly.

#### Parameters

<i>Fildes</i>	Specifies the file descriptor identifying the open file of the part to clear.
<i>Offset</i>	Specifies where to begin clearing the file.
<i>Length</i>	Specifies the number of bytes to be cleared.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an `errno` value defined below.

#### Error Conditions

<code>EBADF</code>	The specified file descriptor does not correspond to a file opened for writing.
--------------------	---

EBUSY	The specified file descriptor is currently in use.
EINVAL	The <i>Length</i> or <i>Offset</i> argument is invalid.

#### See Also

**hpss\_Fclear.**

#### Notes

None.

## 2.1.29. hpss\_Fclose

#### Purpose

Close an HPSS Stream.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Fclose(
    HPSS_FILE *hpss_Stream);          /* IN */
```

#### Description

The **hpss\_Fclose** routine flushes any buffered data to the open HPSS stream, then closes the HPSS file.

#### Parameters

*hpss\_Stream*                      The open HPSS Stream pointer.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EBADF	Invalid file descriptor or file not open.
EFAULT	Invalid or NULL pointer.
EBUSY	HPSS file table entry busy.
ESTALE	Stale connection.
EIO	Internal error condition.

#### See Also

**hpss\_Fflush**

#### Notes

None

## 2.1.30. hpss\_Fcntl

### Purpose

Control HPSS open file descriptors.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Fcntl(
    int    hpss_fd,          /* IN */
    int    Cmd,              /* IN */
    long    Arg);            /* IN */
```

### Description

This function will accept the following commands in the *Cmd* argument and perform the indicated actions:

F_GETFL	Get the O_NONBLOCK flag for an open file.
F_SETFL	Set the O_NONBLOCK flag for an open file.
F_HPSS_SET_COS	Set the the class of service for the open file.
F_HPSS_SET_FILE_SIZE	Set the open HPSS file to the appropriate COS based on size of file given by 'Arg'.

### Parameters

<i>hpss_fd</i>	The open HPSS file descriptor.
<i>Cmd</i>	The input command to be invoked (see above).
<i>Arg</i>	The argument of the 'Cmd' parameter: F_GETFL/F_SETFL - N/A F_HPSS_SET_COS - The COS identifier F_HPSS_SET_FILE_SIZE - Size of file in bytes.

### Return Values

If the F\_GETFL/F\_SETFL command is successful, the file's O\_NONBLOCK flag will be returned. For the other commands, a zero will be returned if the command is successful. Otherwise, a negative value will be returned indicating the error encountered:

### Error Conditions

EBADF	Invalid file descriptor or file not open.
EFAULT	Invalid object handle.
EBUSY	HPSS file table entry busy.
ESTALE	Stale Connection.
ENOTSUP	Command not supported.

### See Also



None

#### Notes

None

## 2.1.31. `hpss_Fflush`

### Purpose

Flush an HPSS stream.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Fflush(
    HPSS_FILE    *hpss_Stream);          /* IN */
```

### Description

The '`hpss_Fflush`' function writes any buffered data for the stream specified by the `hpss_Stream` parameter, leaving the stream open.

### Parameters

*hpss\_Stream*                      The open HPSS stream pointer.

### Return Values

If the operation is successful, a zero value is returned. Otherwise, a negative value is returned whose absolute value indicates the error as listed below.

### Error Conditions

EBADF	Invalid file descriptor or file not open.
EINVAL	Invalid or NULL input parameter.
EBUSY	HPSS file table entry busy.
ESTALE	Stale Connection.
EIO	An internal error has occurred.

### See Also

`hpss_Fopen`, `hpss_Fclose`, `hpss_Fread`, `hpss_Fwrite`, `hpss_Fseek`, `hpss_Ftell`

### Notes

None

## 2.1.32. `hpss_Fgetc`

### Purpose

Read a single character from an open file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Fgetc(
    HPSS_FILE*          *stream)    /* IN */
```

## Description

The **hpss\_Fgetc** function returns the next character in the open file *stream*.

## Parameters

*Stream*                      The open HPSS Stream pointer.

## Return Values

Upon successful completion, the value of the next character in the file is returned. EOF is returned if the end of the file is reached. The global `errno` value will be set to a non-zero value if an error condition occurs.

## Error Conditions

EBADF	Invalid file descriptor or file not open.
EINVAL	Invalid input parameter ( <i>Ptr</i> Null or <i>Size</i> zero)
EFAULT	NULL stream pointer.
EBUSY	HPSS file table entry busy.
ESTALE	Stale Connection.
EPERM	File opened for write only or trying to read after a write without a file positioning operation.

## See Also

**hpss\_Fseek, hpss\_Fflush, hpss\_Fwrite, hpss\_Fopen, hpss\_Fclose, hpss\_Ftell, hpss\_Fgets**

## Notes

The behavior of this function is modeled after the system **fgetc()** routine.

## 2.1.33. hpss\_Fgets

### Purpose

Read a string from an open stream.

### Synopsis

```
#include "hpss_api.h"
char*
hpss_Fgets(
    char*          s,          /* IN/OUT */
    int            n,          /* IN */
    HPSS_FILE*     *stream)    /* IN */
```

## Description

The **hpss\_Fgets** function returns the a string from the open file *stream*, up to n characters or a newline.

#### Parameters

<i>s</i>	The buffer for the string data.
<i>n</i>	The maximum number of bytes to read.
<i>Stream</i>	The open HPSS Stream pointer.

#### Return Values

Upon successful completion, a string of n or less bytes is returned. **hpss\_Fgets** will stop return a string of less than n bytes when a newline is reached. If EOF is reached, NULL is returned.

#### Error Conditions

EBADF	Invalid file descriptor or file not open.
EINVAL	Invalid input parameter ( <i>Ptr</i> Null or <i>Size</i> zero)
EFAULT	NULL stream pointer.
EBUSY	HPSS file table entry busy.
ESTALE	Stale Connection.
EPERM	File opened for write only or trying to read after a write without a file positioning operation.
EIO	More than n bytes were read.

#### See Also

**hpss\_Fseek**, **hpss\_Fflush**, **hpss\_Fwrite**, **hpss\_Fopen**, **hpss\_Fclose**, **hpss\_Ftell**, **hpss\_Fgetc**

#### Notes

The behavior of this function is modeled after the system **fgets()** routine.

## 2.1.34. hpss\_FileGetAttributes

#### Purpose

Get attributes for a file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_FileGetAttributes(
    char          *Path,          /* IN */
    hpss_fileattr_t *AttrOut ); /* OUT */
```

#### Description

The **hpss\_FileGetAttributes** function returns the file attribute structure for the file named by *Path*. The attributes are returned in the structure pointed to by *AttrOut*.

#### Parameters

<i>Path</i>	Points to the path name of the file being queried.
<i>AttrOut</i>	Points to the structure that will hold the file attributes.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string..
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_FileGetAttributesHandle**, **hpss\_FileSetAttributes**, **hpss\_Stat**, **hpss\_Fstat**, **hpss\_Lstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

### Notes

The **hpss\_FileGetAttributes** function reads through symbolic links and junctions. It cannot be used to get the attributes of the symbolic link or junction itself.

## 2.1.35. hpss\_FileGetAttributesHandle

### Purpose

Get attributes for a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FileGetAttributesHandle(
    ns_ObjHandle_t      *ObjHandle,      /* IN */
    char                *Path,           /* IN */
    sec_cred_t          *Ucred,          /* IN */
    hpss_authz_token_t   *AuthzTicket,    /* OUT */
    hpss_fileattr_t      *AttrOut );      /* OUT */
```

### Description

The **hpss\_FileGetAttributesHandle** function can be used to query attributes on an entry in the name or file system that is referred to by *ObjHandle* and *Path*. The file attributes are returned in the *AttrOut* parameter.

### Parameters

<i>ObjHandle</i>	Pointer to parent object.
<i>Path</i>	Points to the path name of the file being queried.

<i>Ucred</i>	User credentials.
<i>AuthzTicket</i>	Authorization ticket.
<i>AttrOut</i>	Points to the structure that will hold the file attributes.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

## Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	<i>ObjHandle</i> is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_FileGetAttributes**, **hpss\_FileSetAttributes**, **hpss\_Stat**, **hpss\_Fstat**, **hpss\_Lstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

## Notes

None.

## 2.1.36. hpss\_FileGetAttributesSOID

### Purpose

Get attributes for an HPSS bitfile.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FileGetAttributesSOID(
    hpssoid_t      *BitfileID,      /* IN */
    char           *Path,           /* IN */
    hpss_fileattr_t *AttrOut );     /* OUT */
```

### Description

The **hpss\_FileGetAttributesSOID** function can be used to query attributes on an entry in the name or file system that is referred to by *BitfileID*. It returns one of possibly multiple path names to the bitfile and the file attributes structure for the bitfile.

### Parameters

<i>BitfileID</i>	Pointer to bitfile ID.
------------------	------------------------

<i>Path</i>	Pointer to the path name of the bitfile being queried.
<i>AttrOut</i>	Pointer to the structure that will hold the file attributes.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	<i>BitfileID</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileSetAttributes**, **hpss\_Stat**, **hpss\_Fstat**, **hpss\_Lstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

### Notes

None.

## 2.1.37. hpss\_FileGetXAttributes

### Purpose

Get extended attributes for a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FileGetXAttributes(
    char            *Path,           /* IN */
    unsigned32      Flags,           /* IN */
    unsigned32      StorageLevel,    /* IN */
    hpss_xfileattr_t *AttrOut );    /* OUT */
```

### Description

The **hpss\_FileGetXAttributes** function returns the file extended attribute structure for the file named by *Path*. The file may currently be open, but is not required to be open. The attributes are returned in the structure pointed to by *AttrOut*.

### Parameters

<i>Path</i>	Points to the pathname of the file being queried.
-------------	---

<i>Flags</i>	Specifies the flag that indicates the behavior of the call. The acceptable values are:  API_GET_STATS_FOR_LEVEL - Returns bitfile attributes at the storage level specified by the <i>StorageLevel</i> argument.  API_GET_STATS_FOR_1STLEVEL - Returns bitfile attributes at the first storage level whether or not it contains bitfile data.  API_GET_STATS_FOR_OPTIMIZE - Returns only StripeWidth and OptimumAccessSize for storage level zero. API_GET_STATS_FOR_ALL_LEVELS - Returns bitfile attributes across all storage class levels.
<i>StorageLevel</i>	Specifies the specific storage level to query when the API_GET_STATS_FOR_LEVEL flag is used.
<i>AttrOut</i>	Points to the structure that will hold the file attributes.

### Return Values

Upon successful completion, a value of zero is returned . Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	<i>BitfileID</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_FileSetAttributes, hpss\_FileSetAttributesHandle, hpss\_FileSetAttributesSOID, hpss\_Stat, hpss\_Fstat, hpss\_Lstat, hpss\_GetListAttrs, hpss\_ReadAttrs.**

### Notes

The **hpss\_FileGetXAttributes** function reads through symbolic links and junctions and cannot be used to get the attributes of the symbolic link or junction itself.

This call allocates memory for the returned physical volume conformant array. After the successful completion of this call, the memory should be freed using code similar to the example below.

```
for(i=0;i<HPSS_MAX_STORAGE_LEVELS;i++)
{
    for(j=0;j<AttrOut.SCAattrib[i].NumberOfVVs;j++)
    {
        if (AttrOut.SCAattrib[i].VVAttrib[j].PVList != NULL)
```

```

    {
        free(AttrOut.SCAAttr[i].VVAAttr[j].PVList);
    }
}
}

```

## 2.1.38. hpss\_FileSetAttributes

### Purpose

Alter file attribute values.

### Synopsis

```

#include "hpss_api.h"
int
hpss_FileSetAttributes(
    char                *Path,          /* IN */
    hpss_fileattrbits_t SelFlags,      /* IN */
    hpss_fileattr_t     *AttrIn,       /* IN */
    hpss_fileattr_t     *AttrOut );    /* OUT */

```

### Description

The **hpss\_FileSetAttributes** function changes file attributes for the file named by *Path*, based on the attributes in the structure pointed to by *AttrIn*. The updated file attributes after the completion of the request are returned in the structure pointed to by *AttrOut*.

### Parameters

<i>Path</i>	Points to the name of the file for which attribute values are to be changed.
<i>SelFlags</i>	Bitmask which indicates which attributes are to be set in the Core Server attributes. The following is a list of the file attributes that can be set using this function.
	CORE_ATTR_ACCOUNT
	CORE_ATTR_COMMENT
	CORE_ATTR_COMPOSITE_PERMS
	CORE_ATTR_COS_ID
	CORE_ATTR_DATA_LENGTH
	CORE_ATTR_DM_DATA_STATE_FLAGS
	CORE_ATTR_DM_HANDLE
	CORE_ATTR_DM_HANDLE_LENGTH
	CORE_ATTR_DONT_PURGE
	CORE_ATTR_ENTRY_COUNT
	CORE_ATTR_EXTENDED_ACLS
	CORE_ATTR_FAMILY_ID
	CORE_ATTR_FILESET_HANDLE
	CORE_ATTR_FILESET_ID
	CORE_ATTR_FILESET_ROOT_ID
	CORE_ATTR_FILESET_STATE_FLAGS
	CORE_ATTR_FILESET_TYPE



CORE\_ATTR\_GATEWAY\_UUID  
 CORE\_ATTR\_GIDCORE\_ATTR\_GROUP\_PERMS  
 CORE\_ATTR\_LINK\_COUNT  
 CORE\_ATTR\_MAC\_SEC\_LABEL  
 CORE\_ATTR\_OPEN\_COUNT  
 CORE\_ATTR\_OTHER\_PERMS  
 CORE\_ATTR\_READ\_COUNT  
 CORE\_ATTR\_REALM\_ID  
 CORE\_ATTR\_REGISTER\_BITMAP  
 CORE\_ATTR\_SET\_GIDCORE\_ATTR\_SET\_STICKY  
 CORE\_ATTR\_SET\_UID  
 CORE\_ATTR\_SUB\_SYSTEM\_ID  
 CORE\_ATTR\_TIME\_CREATED  
 CORE\_ATTR\_TIME\_LAST\_READ  
 CORE\_ATTR\_TIME\_LAST\_WRITTEN  
 CORE\_ATTR\_TIME\_MODIFIED  
 CORE\_ATTR\_TYPE  
 CORE\_ATTR\_UID  
 CORE\_ATTR\_USER\_PERMS  
 CORE\_ATTR\_WRITE\_COUNT

*AttrIn* Points to a structure containing the new attribute values.

*AttrOut* Points to a structure that will contain the file attribute values after completion of this request.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

## Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> , <i>AttrIn</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	An attribute value or selection flag is invalid.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOSPC	Resources could not be allocated to satisfy the request.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EOPNOTSUPP	The requested change is not supported.
EPERM	The client does not have the appropriate privileges to change the file's attributes.

## See Also

**hpss\_FileGetAttributes, hpss\_FileGetAttributesHandle, hpss\_FileGetAttributesSOID, hpss\_FileSetAttributesHandle, hpss\_FileSetAttributesSOID, hpss\_Chown, hpss\_Chmod, hpss\_Utime.**

## Notes

1. The Bitfile ID cannot be set using this function.
2. The Account Code can be set only if site-style accounting is being used.
3. A non-gateway client cannot specify both the Account Code and UID on the same call even if site-style accounting is being used.
4. The contents of *AttrOut* will be zero upon return for a gateway client as the DMAP Gateway doesn't return any attributes.

## 2.1.39. hpss\_FileSetAttributesHandle

### Purpose

Alter file attribute values.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FileSetAttributesHandle(
    ns_ObjHandle_t      ObjHandle, /* IN */
    char                *Path,     /* IN */
    sec_cred_t          *Ucred,    /* IN */
    hpss_fileattrbits_t SelfFlags, /* IN */
    hpss_fileattr_t     *AttrIn,   /* IN */
    hpss_fileattr_t     *AttrOut ); /* OUT */
```

### Description

The **hpss\_FileSetAttributesHandle** function can be used to change attributes on an entry in the name or file system that is referred to by *Path* and *ObjHandle*.

### Parameters

<i>ObjHandle</i>	Parent object handle.
<i>Path</i>	Points to the name of the file for which attribute values are to be changed.
<i>Ucred</i>	User credentials.
<i>SelfFlags</i>	Bitmask which indicates which attributes are to be set in the Core Server attributes. Refer to <b>hpss_FileSetAttributes</b> for a list of attributes that can be set using this function.
<i>AttrIn</i>	Points to a structure containing the new attribute values.
<i>AttrOut</i>	Points to a structure that will contain the file attribute values after completion of this request.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
--------	---

EFAULT	The <i>Path</i> , <i>AttrIn</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	An attribute value or selection flag is invalid, or the <i>ObjHandle</i> is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOSPC	Resources could not be allocated to satisfy the request.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EOPNOTSUPP	The requested change is not supported.
EPERM	The client does not have the appropriate privileges to change the file's attributes.

#### See Also

**hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileSetAttributesHandle**, **hpss\_FileSetAttributesSOID**, **hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**.

#### Notes

1. The Bitfile ID cannot be set using this function.
2. The Account Code can be set only if site-style accounting is being used.
3. A non-gateway client cannot specify both the Account Code and UID on the same call even if site-style accounting is being used.
4. The contents of *AttrOut* will be zero on return for a gateway client as the DMAP Gateway doesn't return any attributes.

## 2.1.40. hpss\_FileSetAttributesSOID

#### Purpose

Alter file attribute values.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_FileSetAttributesSOID(
    hpssoid_t          *BitfileID, /* IN */
    hpss_fileattrbits_t SelfFlags, /* IN */
    hpss_fileattr_t     *AttrIn,    /* IN */
    hpss_fileattr_t     *AttrOut,   /* OUT */
    char                *Path);     /* OUT */
```

#### Description

The **hpss\_FileSetAttributesSOID** function can be used to set attributes on an entry in the name or file system that is referred to by *BitfileID*. It returns one of possibly multiple path names to the bitfile in *Path* and the file attributes for the entry in *AttrOut*.

#### Parameters

<i>BitfileID</i>	<i>BitfileID</i>
<i>Path</i>	Points to the name of the file for which attribute values are to be changed.
<i>SelFlags</i>	Bitmask which indicates which attributes are to be set in the Core Server attributes. Refer to <b>hpss_FileSetAttributes</b> for a list of attributes that can be set using this function.
<i>AttrIn</i>	Points to a structure containing the new attribute values.
<i>AttrOut</i>	Points to a structure that will contain the file attribute values after completion of this request.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	An attribute value or selection flag is invalid, or <i>BitfileID</i> is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOSPC	Resources could not be allocated to satisfy the request.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EOPNOTSUPP	The requested change is not supported.
EPERM	The client does not have the appropriate privileges to change the file's attributes.

### See Also

**hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileSetAttributesHandle**, **hpss\_FileSetAttributesSOID**, **hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**.

### Notes

1. The Bitfile ID cannot be set using this function.
2. The Account Code can be set only if site-style accounting is being used.
3. A non-gateway client cannot specify both the Account Code and UID on the same call even if site-style accounting is being used.
4. The contents of *AttrOut* will be zero on return for a gateway client as the DMAP Gateway doesn't return any attributes.

## 2.1.41. hpss\_FilesetCreate

### Purpose

Create an HPSS fileset.

## Synopsis

```
#include "hpss_api.h"
int
hpss_FilesetCreate(
    hpss_uuid_t          *CoreServerUUID,      /* IN */
    unsigned32           CreateOptions         /* IN */
    ns_FilesetAttrBits_t FilesetAttrBits,      /* IN */
    ns_FilesetAttrs_t    *FilesetAttrs,        /* IN */
    hpss_AttrBits_t      ObjectAttrBits,       /* IN */
    hpss_Attrs_t          *ObjectAttrs,        /* IN */
    dmg_fileset_info_t    *DMGFSInfo,          /* IN */
    ns_FilesetAttrBits_t  RetFilesetAttrBits,   /* IN */
    hpss_AttrBits_t      RetObjectAttrBits,     /* IN */
    ns_FilesetAttrs_t     *RetFilesetAttrs,     /* OUT */
    hpss_Attrs_t          *RetObjectAttrs,      /* OUT */
    ns_ObjHandle_t        *FilesetHandle );     /* OUT */
```

## Description

The **hpss\_FilesetCreate** function is called to create a new HPSS fileset. A handle to the newly created fileset is returned in the memory pointed to by FilesetHandle.

## Parameters

<i>CoreServerUUID</i>	Points to the Core Server uuid to be used for the create.
<i>CreateOptions</i>	Specifies what type of fileset to create. There are three options: CORE_NS_FILESET - Create the Core Server fileset metadata CORE_DMG_FILESET - Create the DMAP Gateway fileset metadata CORE_BOTH_FILESETS - Create both DMG and Core Server fileset metadata
<i>FilesetAttrBits</i>	Specifies which fileset attributes are to be set.
<i>FilesetAttrs</i>	Points to the fileset attributes to be set.
<i>ObjectAttrBits</i>	Specifies which object attributes are to be set.
<i>ObjectAttrs</i>	Points to the object attributes to be set.
<i>DMGFSInfo</i>	DMAP Gateway fileset information. This should be NULL for HPSS-only filesets. For linked filesets (archived), this is a pointer to a structure containing all the details of the fileset. Refer to the definition of <code>dmg_fileset_info_t</code> for details (this resides in <code>include/dmg/dmg_types.idl</code> ).
<i>RetFilesetAttrBits</i>	Specifies which fileset attributes were set.
<i>RetObjectAttrBits</i>	Specifies which object attributes were set.
<i>RetFilesetAttrs</i>	Points to the fileset attributes that were set.
<i>RetObjectAttrs</i>	Points to the object attributes that were set.
<i>FilesetHandle</i>	Points to the fileset handle created.

## Return values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned; the absolute value of that returned is equal to an errno value.

## Error Conditions

EACCES	The user is not the root user or a trusted user.
EFAULT	The CoreServerUUID, FilesetHandle, FilesetAttrs, ObjectAttrs, RetFilesetAttrs or the RetObjectAttrs is NULL.
EINVAL	The file attributes or attributes bits are invalid.
EEXIST	A file already exist with the specified identifier.

## See Also

**hpss\_FilesetDelete**, **hpss\_FilesetGetAttributes**, **hpss\_FilesetSetAttributes**.

## Notes

1. If *CreateOptions* requests a DMG fileset, then *DMGFSInfo* cannot be NULL. If so, EINVAL will be returned.

## 2.1.42. hpss\_FilesetDelete

### Purpose

Delete an HPSS fileset.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FilesetDelete(
    char            *Name,           /* IN */
    u_signed64      *FilesetId,      /* IN */
    ns_ObjHandle_t  *FilesetHandle ); /* IN */
```

### Description

The **hpss\_FilesetDelete** function is called to delete an existing HPSS fileset by either name, id or handle. A fileset can be identified by either a name, an ID, or the handle to its root. Only one type of identifier can be specified. The other values must be NULL pointers.

### Parameters

<i>Name</i>	Specifies the name of the fileset to be deleted.
<i>FilesetId</i>	Specifies the id of the fileset to be deleted.
<i>FilesetHandle</i>	Specifies the handle of the fileset to be deleted.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned; the absolute value of that returned is equal to an errno value.

### Error Conditions

EACCES	The user is not the root user or a trusted user.
EFAULT	The <i>Name</i> , <i>FilesetId</i> and <i>FilesetHandle</i> arguments are all NULL pointers.
EINVAL	More than one type of fileset identifier was specified.
ENOENT	The specified fileset does not exist.

#### See Also

**hpss\_FilesetCreate**, **hpss\_FilesetGetAttributes**, **hpss\_FilesetSetAttributes**.

#### Notes

None.

## 2.1.43. hpss\_FilesetGetAttributes

#### Purpose

Get attributes for an HPSS fileset.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_FilesetGetAttributes(
    char                *FilesetName,      /* IN */
    u_signed64          *FilesetId,        /* IN */
    ns_ObjHandle_t       *FilesetHandle,    /* IN */
    hpss_uuid_t          *CoreServerUUID,   /* IN */
    ns_FilesetAttrBits_t FilesetAttrBits,   /* IN */
    ns_FilesetAttrs_t    *FilesetAttrs );   /* OUT */
```

#### Description

The **hpss\_FilesetGetAttributes** function is called to retrieve the attribute for a specified HPSS fileset by supplying either a name, id or handle. Only one type of identifier can be specified. The other values must be NULL pointers. If NULL is specified for the Core Server UUID, then the local Core Server will be contacted; otherwise the specified Core Server will be contacted to retrieve the fileset information.

#### Parameters

<i>FilesetName</i>	Specifies the name of the fileset to retrieve attributes for.
<i>FilesetId</i>	Specifies the id of the fileset to retrieve attributes for.
<i>FilesetHandle</i>	Specifies the handle of the fileset to retrieve attributes for.
<i>CoreServerUUID</i>	The Core Server to contact managing the fileset information.
<i>FilesetAttrBits</i>	Specifies the fileset attribute bits that specify the fileset attributes to retrieve.
<i>FilesetAttrs</i>	Points to the returned fileset attributes.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned; the absolute value of that returned is equal to an errno value.

## Error Conditions

EACCES	The user is not the root user or a trusted user.
EFAULT	The <i>Name</i> , <i>FilesetID</i> and <i>FilesetHandle</i> arguments are all NULL pointers.
EINVAL	More than one type of fileset identifier was specified or invalid file set attribute bits were specified.
ENOENT	The specified fileset does not exist.

## See Also

**hpss\_FilesetSetAttributes**, **hpss\_FilesetCreate**, **hpss\_FilesetDelete**.

## Notes

None.

## 2.1.44. hpss\_FilesetListAll

### Purpose

Obtain a list of all the HPSS filesets.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FilesetListAll(
    u_signed64          OffsetIn,          /* IN */
    unsigned32          Entries,          /* IN */
    unsigned32          *End,             /* OUT */
    u_signed64          *OffsetOut,        /* OUT */
    hpss_global_fsent_t *FSentPtr );      /* OUT */
```

### Description

The **hpss\_FilesetListAll** routine is called to get the global fileset attributes for all the filesets in the HPSS site.

### Parameters

<i>OffsetIn</i>	The offset of the first fileset entry to be read. This should be set to zero to start before the first call, and subsequent entries can be read by providing the value returned in <i>OffsetOut</i> .
<i>Entries</i>	The number of the fileset entries referenced by <i>FSentPtr</i> in <i>hpss_global_fsent_t</i> entries, for which you have to allocate space.
<i>End</i>	Pointer to an area to contain an indication of whether the last fileset entry is included in the returned list. The value of <i>End</i> specifies if the end of the list was encountered before all the entries were accumulated.
<i>OffsetOut</i>	Pointer to an area to contain the offset of the next fileset entry following those returned by this call. <i>OffsetOut</i> is the location where the lookup stops when it has accumulated the specified number of fileset entries.
<i>FSentPtr</i>	Pointer to area to contain returned fileset entries.



## Return Values

Upon successful completion, a nonnegative value indicating the number of fileset entries is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EFAULT	The <i>End</i> , <i>OffsetOut</i> or <i>FSentPtr</i> parameter is a NULL pointer.
EINVAL	The <i>Entries</i> parameter specifies zero entries to read.

## See Also

**hpss\_FilesetCreate.**

## Notes

None.

## 2.1.45. hpss\_FilesetSetAttributes

### Purpose

Set attributes for an HPSS fileset.

### Synopsis

```
#include "hpss_api.h"
int
hpss_FilesetSetAttributes(
    char                *Name,                /* IN */
    u_signed64          *FilesetId,           /* IN */
    ns_ObjHandle_t      *FilesetHandle,       /* IN */
    ns_FilesetAttrBits_t FilesetAttrBitsIn,   /* IN */
    ns_FilesetAttrs_t   *FilesetAttrsIn,     /* IN */
    ns_FilesetAttrBits_t FilesetAttrBitsOut,  /* IN */
    ns_FilesetAttrs_t   *FilesetAttrsOut );  /* OUT */
```

### Description

The **hpss\_FilesetSetAttributes** function is called to set the attribute for a specified Core Server fileset by either name, id or handle. Only one type of identifier can be specified. The other values must be NULL pointers.

### Parameters

<i>Name</i>	Specifies the name of the fileset to retrieve attributes for.
<i>FilesetId</i>	Specifies the id of the fileset to retrieve attributes for.
<i>FilesetHandle</i>	Specifies the handle of the fileset to retrieve attributes for.
<i>FilesetAttrBits</i>	Specifies the fileset attribute bits that specify the fileset attribute values that are to be set.
<i>FilesetAttrs</i>	Points to the fileset attribute values to be set.

<i>FilesetAttrBitsOut</i>	Specifies the fileset attribute bits that specify the fileset attribute values that are to be returned.
<i>FilesetAttrsOut</i>	Points to the returned fileset attribute values.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned; the absolute value of that returned is equal to an errno value.

### Error Conditions

EACCES	The user is not the root user or a trusted user.
EFAULT	The <i>Name</i> , <i>FilesetID</i> and <i>FilesetHandle</i> arguments are all NULL pointers.
EINVAL	More than one type of fileset identifier was specified or invalid file set attributes (or attribute bits) were specified.
ENOENT	The specified fileset does not exist.

### See Also

**hpss\_FilesetGetAttributes**, **hpss\_FilesetCreate**, **hpss\_FilesetDelete**.

### Notes

A fileset can be identified by either a name, an ID, or the handle to its root. Only one type of identifier is valid. If more than one of the fileset identifiers have non-null pointers, then the call will fail with an EINVAL.

## 2.1.46. hpss\_Fopen

### Purpose

Open an HPSS file stream.

### Synopsis

```
#include "hpss_api.h"
HPSS_FILE *
hpss_Fopen(
    char *path,          /* IN */
    char *mode);         /* IN */
```

### Description

This function will open an HPSS file pointed to by the path parameter, associate a stream with it and return a pointer to an HPSS\_FILE structure. The pointer can be used with any of the HPSS stream functions.

### Parameters

<i>path</i>	Character string specifying the path/name of the HPSS file to be opened.
<i>mode</i>	File mode (r,w,a,r+,w+,a+).

### Return Values

If the open command is successful, a pointer to the HPSS\_FILE structure will be returned. Otherwise, a

NULL pointer will be returned and the system errno value will be set to indicate the error.

#### Error Conditions

EFAULT	Path parameter is NULL.
EINVAL	Invalid mode parameter.
EIO	Internal error.
ENOENT	Path parameter points to an empty string.
ENOMEM	Memory allocation failure.
EMFILE	Too many open files.

#### See Also

**hpss\_Fclose, hpss\_Fread, hpss\_Fwrite, hpss\_Fseek, hpss\_Ftell**

#### Notes

If the file is open for update, an output operation cannot be directly followed by an input unless an intervening file positioning call is made. Also, an input operation cannot be directly followed by an output operation unless an intervening file positioning call is made except when the input encounters an EOF.

Following is a list of valid values for the mode parameter and their effects on the HPSS stream:

<b>r</b>	Open text file for reading. The stream is positioned at the beginning of the file.
<b>r+</b>	Open for reading and writing. The stream is positioned at the beginning of the file.
<b>w</b>	Truncate the file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
<b>w+</b>	Open for reading and writing. The file is created if it doesn't exist, otherwise it is truncated to zero length. The stream is positioned at the beginning of the file.
<b>a</b>	Open for appending (writing at end of file). The file is created if it doesn't exist. The stream is positioned at the end of the file.
<b>a+</b>	Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.

## 2.1.47. hpss\_Fpreallocate

#### Purpose

Set the length of a file and preallocate storage segments.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Fpreallocate(
    int          Fildes,      /* IN */
    unsigned64   Length );    /* IN */
```

## Description

The **hpss\_Fpreallocate** routine sets the length of an open file, specified by the *Fildes* argument. The *Length* parameter specifies the requested length. It must be greater than the current size of the file. Additional storage space is preallocated for the file and a hole is created in the file from the current size to the requested length.

## Parameters

<i>Fildes</i>	Specifies file descriptor identifying file to be queried.
<i>Length</i>	Specifies the desired length of the file.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	The specified file descriptor does not correspond to a file opened for writing.
EBUSY	The specified file descriptor is currently in use.
ENOSPC	The requested storage resources could not be allocated.
EINVAL	There is not a disk storage class at the top of the storage hierarchy.

## See Also

**hpss\_Ftruncate, hpss\_Truncate, hpss\_Fclear, hpss\_FileSetAttributes.**

## Notes

There must be a disk storage class at the top of the storage hierarchy in which the file resides.

## 2.1.48. hpss\_Fread

### Purpose

Read from an HPSS Stream.

### Synopsis

```
#include "hpss_api.h"
size_t
hpss_Fread(
    void          *Ptr          /* IN */
    size_t        Size,         /* IN */
    size_t        Num,          /* IN */
    HPSS_FILE     *hpss_Stream) /* IN */;
```

### Description

The **hpss\_Fread** function will provide a buffered I/O front-end to the **hpss\_Read** function. It will support both 32-bit and 64-bit `size_t` values depending on how the library is built.

The function will copy the number of data items specified by the '*Num*' parameter into an array pointed to

by the '*Ptr*' parameter from the input stream *hpss\_Stream*. Each data item consists of the number of bytes specified by the '*Size*' parameter.

The function will stop copying bytes if an end-of-file (EOF) or error condition is encountered while reading from the input specified by the '*hpss\_Stream*' parameter, or the number of data items specified by the '*Num*' parameter have been copied.

#### Parameters

<i>Ptr</i>	Pointer to an array of data items of size ' <i>Size</i> ' bytes.
<i>Size</i>	Specifies the size in bytes of each data item in the ' <i>Ptr</i> ' array.
<i>Num</i>	Number of data items in the ' <i>Ptr</i> ' array.
<i>hpss_Stream</i>	The open HPSS Stream pointer.

#### Return Values

#### Return Values

On successful completion, the number of data items read will be returned. The global `errno` value will be set to a non-zero value if an error condition occurs.

#### Error Conditions

EBADF	Invalid file descriptor or file not open.
EINVAL	Invalid input parameter ( <i>Ptr</i> Null or <i>Size</i> zero)
EFAULT	NULL stream pointer.
EBUSY	HPSS file table entry busy.
ESTALE	Stale Connection.
EPERM	File opened for write only or trying to read after a write without a file positioning operation.

#### See Also

**hpss\_Fseek, hpss\_Fflush, hpss\_Fwrite, hpss\_Fopen, hpss\_Fclose, hpss\_Ftell**

#### Notes

The behavior of this function is modeled after the system **fread()** routine.

## 2.1.49. hpss\_Fseek

#### Purpose

Seek on an HPSS Stream.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Fseek (
    HPSS_FILE      *hpss_Stream,          /* IN */
    ssize_t         OffsetIn,              /* IN */
    int             Whence) ;              /* IN */
```

## Description

The **hpss\_Fseek** routine sets the file position of an opened HPSS file.

## Parameters

<i>hpss_Stream</i>	The open HPSS Stream pointer.
<i>OffsetIn</i>	The number of bytes to adjust the HPSS file position. Can be negative or positive, or zero.
<i>Whence</i>	Origin for the seek.  SEEK_SET      File position set to OffsetIn from beginning of file. SEEK_CUR      File position adjusted by OffsetIn from current file position. SEEK_END      File position set to file size + OffsetIn.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	Invalid file descriptor or file not open.
EINVAL	Invalid input parameter.
EFAULT	Invalid or NULL pointer.
EBUSY	HPSS file table entry busy.
ESTALE	Stale Connection.
EPERM	File opened for write only or trying to read after a write without a file positioning operation.

## See Also

**hpss\_Lseek**, **hpss\_Fflush**

## Notes

The behavior of this function is modeled after the system **fseek()** routine.

## 2.1.50. hpss\_Fstat

### Purpose

Get file status (POSIX).

### Synopsis

```
#include "hpss_api.h"
int
hpss_Fstat(
    int          Fildes,      /* IN */
    hpss_stat_t *Buf );      /* OUT */
```

## Description

The **hpss\_Fstat** function obtains information about the open file identified by *Fildes* and returns it in the structure pointed to by *Buf*. Refer to POSIX.1 for more detailed information.

## Parameters

<i>Fildes</i>	Specifies the file descriptor identifying the file to be queried.
<i>Buf</i>	Points to a stat structure that will contain the information for the file.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **stat**.

## Error Conditions

EBADF	The file descriptor supplied does not correspond to an open file.
EBUSY	Another thread is currently manipulating this entry.
EFAULT	The <i>Buf</i> parameter is a NULL pointer.

## See Also

**hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**, **hpss\_FileGetAttributes**, **hpss\_FileSetAttributes**, **hpss\_Stat**, **hpss\_Lstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

## Notes

None.

## 2.1.51. hpss\_Fsync

### Purpose

This function is currently a no-op and will return a zero value as long as *hpss\_fd* is a valid HPSS file descriptor.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Fsync(
    int    hpss_fd);          /* IN */
```

### Description

The **hpss\_Fsync** routine returns a zero as long as the *hpss\_fd* is valid.

### Parameters

<i>hpss_fd</i>	HPSS file descriptor.
----------------	-----------------------

### Return Values

If the input value for *hpss\_fd* is valid, zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	Invalid file descriptor.
-------	--------------------------

## See Also

None

## Notes

None

## 2.1.52. `hpss_Ftell`

### Purpose

Get the current file position of an HPSS file stream.

### Synopsis

```
#include "hpss_api.h"
long
hpss_Ftell(
    HPSS_FILE    *hpss_Stream);    /* IN */
```

### Description

The `hpss_Ftell` routine returns the file position indicator of the open HPSS file.

### Parameters

<i>hpss_Stream</i>	HPSS stream pointer.
--------------------	----------------------

### Return Values

On successful completion the value of the current file position is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an `errno` value defined below.

## Error Conditions

EBADF	Invalid file descriptor.
EFAULT	NULL input pointer.

## See Also

None

## Notes

None

## 2.1.53. `hpss_Ftruncate`

### Purpose

Set the length of an HPSS file stream.

### Synopsis



```
#include "hpss_api.h"
hpss_Ftruncate(
    int          Fildes,      /* IN */
    u_signed64   Length );   /* IN */
```

## Description

The **hpss\_Ftruncate** routine sets the length of an open file, specified by the *Fildes* argument. If the new file length is less than the current length, the space allocated beyond the new length will be freed. If the new length is greater than the current length, a hole is created in the file.

## Parameters

<i>Fildes</i>	Specifies the file descriptor identifying the open file for which the length is to be set.
<i>Length</i>	Specifies the new length of the file.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	The specified file descriptor does not correspond to a file opened for writing.
EBUSY	The specified file descriptor is currently in use.
EINVAL	No DMAP support compiled in.

## See Also

**hpss\_Truncate**, **hpss\_Fclear**, **hpss\_FileSetAttributes**.

## Notes

None.

## 2.1.54. hpss\_Fwrite

### Purpose

Write to an HPSS file stream.

### Synopsis

```
#include "hpss_api.h"
size_t
hpss_Fwrite(
    void          *Ptr,          /* IN */
    size_t        Size,          /* IN */
    size_t        Num,           /* IN */
    HPSS_FILE     *hpss_Stream); /* IN */
```

## Description

The **hpss\_Fwrite** function will provide a buffered I/O front-end to the **hpss\_Write** function. It will

support both 32-bit and 64-bit `size_t` values depending on how the library is built.

The function will write the number of data items specified by the '*Num*' parameter from an array pointed to by the '*Ptr*' parameter to the input stream `hpss_Stream`. Each data item consists of the number of bytes specified by the '*Size*' parameter.

The function will stop writing bytes if an error condition is encountered or the number of data items specified by the '*Num*' parameter have been written.

#### Parameters

<i>Ptr</i>	Pointer to an array of data items of size ' <i>Size</i> ' bytes.
<i>Size</i>	Specifies the size in bytes of each data item in the ' <i>Ptr</i> ' array.
<i>Num</i>	Number of data items in the ' <i>Ptr</i> ' array.
<i>hpss_Stream</i>	HPSS stream pointer.

#### Return Values

On successful completion, the number of data items written will be returned. The global `errno` value will be set to a non-zero value if an error condition occurs.

#### Error Conditions

EBADF	Invalid HPSS file descriptor.
EFAULT	NULL pointer.
EFBIG	Size of write request greater than 2GB.
EBUSY	Another thread is accessing the file table entry.
ENOENT	File doesn't exist.
ENOMEM	Memory allocation failure.
EINVAL	Invalid input parameter.

#### See Also

`hpss_Fread`, `hpss_Fopen`, `hpss_Fclose`, `hpss_Fsync`, `hpss_Fcntl`, `hpss_Ftell`, `hpss_Fflush`

#### Notes

None

## 2.1.55. `hpss_GetAcct`

#### Purpose

Query the default and current account codes.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_GetAcct(
    acct_rec_t *RetDefAcct,    /* OUT */
    acct_rec_t *RetCurAcct); /* OUT */
```

## Description

The **hpss\_GetAcct** routine returns the default and current account codes for the calling thread.

## Parameters

<i>RetDefAcct</i>	Points to an area that will contain the default account code.
<i>RetCurAcct</i>	Points to an area that will contain the current account code.

## Return Values

Upon successful completion, **hpss\_GetAcct** returns zero. Otherwise, **hpss\_GetAcct** returns a negative value; the absolute value of which indicates the specific error.

## Error Conditions

EFAULT	The <i>RetDefAcct</i> or <i>RetCurAcct</i> parameter is a NULL pointer.
--------	---

## See Also

**hpss\_AcctCodeToName**, **hpss\_AcctNameToCode**, **hpss\_Chacct**, **hpss\_ChacctByName**, **hpss\_GetAcctName**, **hpss\_SetAcct**, **hpss\_SetAcctByName**

## Notes

None.

## 2.1.56. hpss\_GetAcctName

### Purpose

Retrieve the current account name.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetAcctName (
    char *AcctName ) /* OUT */
```

### Description

The **hpss\_GetAcctName** routine retrieves the name of the current session account for this thread. Since each site contacted by each thread in the Client API can have its own session account name, the account name for the site managing the current working directory is returned.

### Parameters

<i>AcctName</i>	The name of the thread's current session account. Account name should be a string of at least HPSS_MAX_ACCOUNT_NAME (128) characters.
-----------------	---

### Return Values

Upon successful completion, **hpss\_GetAcctName** returns 0. Otherwise, **hpss\_GetAcctName** returns a negative value, the absolute value of which indicates the specific error.

### Error Conditions

EFAULT                      A NULL AcctName was provided.

#### See Also

**hpss\_GetAcct, hpss\_Chacct, hpss\_SetAcct, hpss\_SetAcctByName, hpss\_ChacctByName.**

#### Notes

None.

## 2.1.57. hpss\_GetACL

### Purpose

Query the Access Control List of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetACL(
    char          *Path,          /* IN */
    unsigned32     Options,       /* IN */
    ns_ACLConfArray_t **ACL );   /* OUT */
```

### Description

The **hpss\_GetACL** function returns the access control list information for the named file.

### Parameters

<i>Path</i>	Names the file for which the ACL is being queried.
<i>Options</i>	Bit vector used to specify what type of ACL is to be retrieved. One of:  HPSS_ACL_OPTION_OBJ – return object's normal ACL.  HPSS_ACL_OPTION_IO – return the initial-object ACL. (only valid for directory objects)  HPSS_ACL_OPTION_IC – return the initial-container ACL. (only valid for directory objects)
<i>ACL</i>	Points to the beginning of the returned list of ACL entries.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below:

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	Exactly one of the HPSS_ACL_OPTION_* bits must be set in the <i>Options</i> bit vector to avoid receiving this error.

ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

#### See Also

**hpss\_GetACLHandle, hpss\_SetACL, hpss\_SetACLHandle, hpss\_DeleteACL, hpss\_DeleteACLHandle, hpss\_UpdateACL, hpss\_UpdateACLHandle.**

#### Notes

The user is responsible for freeing the *ACL* return parameter.

## 2.1.58. hpss\_GetACLHandle

#### Purpose

Query the Access Control List of a file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_GetACLHandle(
    ns_ObjHandle_t    *ObjHandle, /* IN */
    char              *Path,      /* IN */
    sec_cred_t        *Ucred,     /* IN */
    unsigned32        Options,    /* IN */
    ns_ACLConfArray_t **ACL );    /* OUT */
```

#### Description

The **hpss\_GetACLHandle** function returns the access control list information for the named file taken relative to the directory indicated by *ObjHandle*.

#### Parameters

<i>ns_ObjHandle_t</i>	Parent object handle.
<i>Path</i>	Names the file for which the ACL is being queried.
<i>Ucred</i>	User credentials.
<i>Options</i>	Bit vector used to specify what type of ACL is to be retrieved. One of: HPSS_ACL_OPTION_OBJ – return object's normal ACL. HPSS_ACL_OPTION_IO – return the initial-object ACL. (only valid for directory objects) HPSS_ACL_OPTION_IC – return the initial-container ACL. (only valid for directory objects)
<i>ACL</i>	Points to the beginning of the returned list of ACL entries.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below:

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	Exactly one of the HPSS_ACL_OPTION_* bits must be set in the <i>Options</i> bit vector to avoid receiving this error, or <i>ObjHandle</i> is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

#### See Also

**hpss\_SetACL, hpss\_GetACL, hpss\_DeleteACL, hpss\_DeleteACLHandle, hpss\_UpdateACL, hpss\_UpdateACLHandle.**

#### Notes

1. The user is responsible for freeing the *ACL* return parameter.
2. If the *Ucred* parameter is NULL, the credentials in the current thread context are used.

## 2.1.59. hpss\_GetAsynchStatus

#### Purpose

Get status associated with a background request where callback information was previously provided.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_GetAsynchStatus (
    signed32      CallbackId, /*IN*/
    hpssoid_t     BfId,       /*IN*/
    signed32      Status );   /*OUT*/
```

#### Description

The **hpss\_GetAsynchStatus** routine queries the Root Core Server to obtain status information for the specified background request.

#### Parameters

<i>CallbackId</i>	Identifies the background request.
<i>BfId</i>	Bitfile identifier returned from the stage callback request.
<i>Status</i>	Returned stage status. This can be one of the following: HPSS_STAGE_STATUS_UNKNOWN - no status available.

HPSS\_STAGE\_STATUS\_QUEUED - request still in queue and not yet active.

HPSS\_STAGE\_STATUS\_ACTIVE - stage active and in progress.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EINVAL                      The *Bfld* parameter is not a valid pointer.

EFAULT                      The *Status* parameter is a NULL pointer.

## See Also

**hpss\_StageCallback**, **hpss\_Stage**.

## Notes

None.

## 2.1.60. hpss\_GetAttrHandle

### Purpose

Get attributes for a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetAttrHandle(
    ns_ObjHandle_t      *ObjHandle,      /* IN */
    char                *Path,           /* IN */
    sec_cred_t          *Ucred,          /* IN */
    ns_ObjHandle_t      *HandleOut,      /* OUT */
    hpss_authz_token_t  *AuthzTicket,    /* OUT */
    hpss_vattr_t        *AttrOut);      /* OUT */
```

### Description

The **hpss\_GetAttrHandle** function obtains information about the file or directory named by 'Path', taken relative to the directory indicated by *ObjHandle*. Attributes are returned in the area pointed to by *AttrOut*. If *Path* refers to a symbolic link, information will be returned about the link itself.

### Parameters

<i>ns_ObjHandle_t</i>	Parent object handle.
<i>Path</i>	Points to the path name of the file being queried.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context will be used.
<i>HandleOut</i>	Returned object handle.
<i>AttrOut</i>	Points to the structure that will hold the file attributes.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	The <i>ObjHandle</i> parameter is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_FileGetAttributesHandle, hpss\_FileGetXAttributes, hpss\_GetRawAttrHandle, hpss\_FileGetAttributesSOID, hpss\_FileSetAttributes, hpss\_Stat, hpss\_Fstat, hpss\_Lstat, hpss\_GetListAttrs, hpss\_ReadAttrs.**

## Notes

None.

## 2.1.61. hpss\_GetAuthType

### Purpose

Get current default authenticator type for the specified authentication mechanism.

### Synopsis

```
#include "hpss_api.h"
signed32
hpss_GetAuthType (
    hpss_authn_mech_t      AuthMech,          /* IN */
    hpss_rpc_auth_type_t   *AuthType) ;      /* OUT */
```

### Description

The **hpss\_GetAuthType** is called to get the current (or default) authenticator type for the specified authentication mechanism.

### Parameter

<i>AuthMech</i>	Authentication mechanism being used. Valid values are: hpss_authn_mech_krb5 (Kerberos 5) hpss_authn_mech_unix (Unix)
<i>AuthType</i>	Authenticator type. Valid values are: hpss_rpc_auth_type_keytab (Kerberos 5) hpss_rpc_auth_type_keyfile (Unix)



### Return Values

Upon successful completion, a valid authenticator type is returned. If the authtype cannot be determined, HPSS\_ENOTSUPPORTED is returned.

### Error Conditions

ENOTSUPPORTED      Authentication mechanism is not supported.

### See Also

None.

### Notes

None.

## 2.1.62. hpss\_GetConfiguration

### Purpose

Query the current Client API configuration information.

### Synopsis

```
#include "hpss_api.h"
long
hpss_GetConfiguration(
    api_config_t      *ConfigOut );      /* OUT */
```

### Description

The **hpss\_GetConfiguration** routine returns the current configuration values for the Client API.

### Parameters

*ConfigOut*      Points to an area that will contain the current configuration attribute value settings.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT      The *ConfigOut* parameter is a NULL pointer.

### See Also

**hpss\_SetConfiguration.**

### Notes

None.

## 2.1.63. hpss\_Getcwd

### Purpose

Get current working directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Getcwd(
    char      *Buf,      /* OUT */
    size_t    Size );   /* IN */
```

### Description

The **hpss\_Getcwd** function copies an absolute path name of the current working directory to the character array pointed to by *Buf*. The *Size* argument is the size in bytes of the array pointed to by *Buf*.

### Parameters

<i>Buf</i>	Points to an array to contain the current working directory path name.
<i>Size</i>	Specifies the size, in bytes, of the array pointed to by <i>Buf</i> .

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **getcwd**.

### Error Conditions

EACCES	Read or Search permission is denied on a component of the path name.
EFAULT	The <i>Buf</i> parameter is a NULL pointer.
EINVAL	The <i>Size</i> argument is zero.
ERANGE	The <i>Size</i> argument is greater than zero, but smaller than the length of the path name plus 1.

### See Also

**hpss\_Chdir**, **hpss\_Chroot**.

### Notes

**hpss\_Getcwd** is altered from POSIX.1 *getcwd* in that it returns an integer value to be more consistent with other HPSS calls.

## 2.1.64. hpss\_GetDistFile

### Purpose

Get a distributed file's information.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetDistFile(
    int          Fildes,      /* IN */
    api_dist_file_info_t *FileInfo); /* OUT */
```

## Description

The **hpss\_GetDistFile** function extracts a file table entry for a supplied open file descriptor. The function returns a pointer to the file table information.

## Parameters

<i>FileDes</i>	Open file descriptor.
<i>FileInfo</i>	Pointer to returned file information.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value.

## Error Conditions

EBADF	Invalid file descriptor <i>Fildes</i> .
EBUSY	Another thread is currently manipulating this entry.
EFAULT	<i>FileInfo</i> is a NULL pointer.

## See Also

**hpss\_InsertDistFile**

## Notes

None.

## 2.1.65. hpss\_GetJunctions

### Purpose

Get a list of all junctions residing in a given storage subsystem's name space.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetJunctions(
    unsigned32      SubsystemID,      /* IN */
    u_signed64      OffsetIn,         /* IN */
    unsigned32      Entries,          /* IN */
    unsigned32      *End,             /* OUT */
    u_signed64      *OffsetOut,       /* OUT */
    hpss_junction_ent_t *JentPtr );  /* OUT */
```

## Description

The **hpss\_GetJunctions** routine is called to get an array of junctions residing in the specified storage subsystem.

## Parameters

<i>SubsystemID</i>	ID of the storage subsystem where the junction resides.
--------------------	---

<i>OffsetIn</i>	The offset of the first junction entry to be read. This should be set to zero to start before the first call, and subsequent entries can be read by providing the value returned in <i>OffsetOut</i> .
<i>Entries</i>	The number of the junction entries referenced by <i>JentPtr</i> in <i>hpss_junction_ent_t</i> entries, for which you have to allocate space.
<i>End</i>	Pointer to an area to contain an indication of whether the last junction entry is included in the returned list. The value of <i>End</i> specifies if the end of the list was encountered before all the entries were accumulated.
<i>OffsetOut</i>	Pointer to an area to contain the offset of the next junction entry following those returned by this call. <i>OffsetOut</i> is the location where the lookup stops when it has accumulated the specified number of junction entries.
<i>JentPtr</i>	Pointer to an area to contain the returned junction entries.

### Return Values

Upon successful completion, a nonnegative value indicating the number of junction entries read is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT	The <i>End</i> , <i>OffsetOut</i> or <i>JentPtr</i> parameter is a NULL pointer.
EINVAL	The <i>Entries</i> parameter specifies zero entries to read.

### See Also

**hpss\_FilesetCreate**, **hpss\_JunctionCreate**, **hpss\_JunctionCreateHandle**, **hpss\_JunctionDelete**, **hpss\_JunctionDeleteHandle**.

### Notes

None.

## 2.1.66. hpss\_GetListAttrs

### Purpose

Get attributes for a file, suitable for a directory listing.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetListAttrs (
    char          *Path,          /* IN */
    hpss_Attrs_t  *AttrOut );    /* OUT */
```

### Description

The **hpss\_GetListAttrs** function returns the attributes associated with the file named by *Path*. The attributes include information suitable for a long directory listing, including 64-bit file length and Class of Service. If *Path* refers to a junction, the junction will be traversed. If *Path* refers to a symbolic link, information will be returned about the link itself.

## Parameters

<i>Path</i>	Points to the path name of the file being queried.
<i>AttrOut</i>	Points to a structure that will contain the attribute information for the file; not to be confused with the stat structure returned by <b>hpss_Stat</b> or <b>hpss_Fstat</b> .

## Return Values

Upon successful completion, a value of zero is returned . Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

## Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**, **hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileSetAttributes**, **hpss\_FileSetAttributesHandle**, **hpss\_FileSetAttributesSOID**, **hpss\_Stat**, **hpss\_Fstat**, **hpss\_ReadAttrs**, **hpss\_GetJunctionAttrs**.

## Notes

None.

## 2.1.67. hpss\_GetJunctionAttrs

### Purpose

Get attributes for a file or junction. It does not traverse a junction or symbolic link.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetJunctionAttrs(
    char            *Path,            /* IN */
    hpss_Attrs_t    *AttrOut );      /* OUT */
```

### Description

The **hpss\_GetJunctionAttrs** function returns the attributes associated with the file named by *Path*. The attributes include information suitable for a long directory listing, including 64-bit file length and Class of Service. If *Path* refers to a junction or symbolic link, information will be returned about the junction or link itself.

### Parameters

<i>Path</i>	Points to the path name of the file being queried.
-------------	--

<i>AttrOut</i>	Points to a structure that will contain the attribute information for the file; not to be confused with the stat structure returned by <b>hpss_Stat</b> or <b>hpss_Fstat</b> .
----------------	--

### Return Values

Upon successful completion, a value of zero is returned . Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_Chown, hpss\_Chmod, hpss\_Utime, hpss\_FileGetAttributes, hpss\_FileGetAttributesHandle, hpss\_FileGetAttributesSOID, hpss\_FileSetAttributes, hpss\_FileSetAttributesHandle, hpss\_FileSetAttributesSOID, hpss\_Stat, hpss\_Fstat, hpss\_ReadAttrs, hpss\_GetListAttrs.**

### Notes

None.

## 2.1.68. hpss\_GetObjId

### Purpose

Get HPSS Object ID given an object handle.

### Synopsis

```
#include "hpss_api.h"
unsigned32
hpss_GetObjId(
    ns_ObjHandle_t    *ObjHandle);    /* IN */
```

### Description

This routine returns the object id given the object handle.

### Parameters

<i>ObjHandle</i>	HPSS object handle.
------------------	---------------------

### Return Values

Upon successful completion, the object id is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT                      The *ObjHandle* parameter is a NULL pointer.

#### See Also

**hpss\_GetObjType**

#### Notes

None.

## 2.1.69. hpss\_GetObjType

### Purpose

Get HPSS Object type given an object handle.

### Synopsis

```
#include "hpss_api.h"
unsigned32
hpss_GetObjType(
    ns_ObjHandle_t    *ObjHandle);    /* IN */
```

### Description

This routine returns the object type given the object handle.

### Parameters

*ObjHandle*                      HPSS object handle.

### Return Values

Upon successful completion, the object type is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below. Valid return types are:

NS\_OBJECT\_TYPE\_DIRECTORY

NS\_OBJECT\_TYPE\_JUNCTION

NS\_OBJECT\_TYPE\_FILE

NS\_OBJECT\_TYPE\_HARD\_LINK

NS\_OBJECT\_TYPE\_SYM\_LINK

### Error Conditions

EFAULT                      The *ObjHandle* parameter is a NULL pointer.

EINVAL                      The *ObjHandle* points to an invalid object.

#### See Also

**hpss\_GetObjId**

#### Notes

None.

## 2.1.70. hpss\_GetPathHandle

### Purpose

Get HPSS pathname and root fileset handle.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetPathHandle(
    ns_ObjHandle_t    *ObjHandle);          /* IN */
    ns_ObjHandle_t    *FilesetRootHandle,   /* OUT */
    char              *Path);              /* OUT */
```

### Description

The **hpss\_GetPathHandle** function outputs a pathname and a root fileset handle that corresponds to an object handle..

### Parameters

<i>ObjHandle</i>	HPSS object handle.
<i>FilesetRootHandle</i>	Fileset root handle.
<i>Path</i>	Path to object identified by <i>ObjHandle</i> parameter.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT	The <i>Path</i> or <i>FilesetRootHandle</i> parameter is NULL.
EINVAL	The <i>ObjHandle</i> is NULL.

### See Also

None.

### Notes

None.

## 2.1.71. hpss\_GetRawAttrHandle

### Purpose

Get attributes for a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_GetRawAttrHandle(
    ns_ObjHandle_t    *ObjHandle,          /* IN */
```



```

char                *Path,                /* IN */
sec_cred_t          *Ucred,              /* IN */
ns_ObjHandle_t      *HandleOut,          /* OUT */
hpss_authz_token_t  *AuthzTicket,       /* OUT */
hpss_vattr_t        *AttrOut);          /* OUT */

```

## Description

The **hpss\_GetRawAttrHandle** function obtains information about the symlink or the junction named by *Path*, taken relative to the directory indicated by *ObjHandle*. Attributes are returned in the area pointed to by *AttrOut*.

## Parameters

<i>ns_ObjHandle_t</i>	Parent object handle.
<i>Path</i>	Points to the path name of the file being queried.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context will be used.
<i>HandleOut</i>	Returned object handle.
<i>AuthzTicket</i>	Returned authorization ticket.
<i>AttrOut</i>	Points to the structure that will hold the file attributes.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

## Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	The <i>ObjHandle</i> parameter is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_FileGetAttributesHandle**, **hpss\_FileGetXAttributes**,  
**hpss\_GetRawAttrHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileSetAttributes**, **hpss\_Stat**,  
**hpss\_Fstat**, **hpss\_Lstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

## Notes

None.

## 2.1.72. hpss\_GetSubSysStats

### Purpose

Query Storage Subsystem statistics.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_GetSubSysStats (
    unsigned32      SubsystemID,      /* IN */
    subsys_stats_t  *StatsOut );      /* OUT */
```

#### Description

The **hpss\_GetSubSysStats** routine returns the stage, migration, purge, and delete counts for a subsystem along with the time these counts were last reset.

#### Parameters

<i>SubsystemID</i>	Subsystem identifier.
<i>StatsOut</i>	Points to an area that will contain the current statistics values.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EFAULT	The <i>StatsOut</i> parameter is a NULL pointer.
--------	--

#### See Also

**hpss\_ResetSubSysStats.**

#### Notes

None.

## 2.1.73. hpss\_GetThreadUcred

#### Purpose

Gets the user credentials for the currently running thread.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_GetThreadUcred (
    sec_cred_t      *RetUcred );      /* OUT */
```

#### Description

The **hpss\_GetThreadUcred** function is used to get the user credentials for the currently running thread.

#### Parameters

<i>RetUcred</i>	On success, this points to the current client credentials.
-----------------	--

## Return Values

Upon successful completion, a value of zero is returned . Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

## Error Conditions

EFAULT *RetUcred* is a NULL pointer.

## See Also

None.

## Notes

None.

## 2.1.74. hpss\_HandleCompare

### Purpose

Compares two object handles.

### Synopsis

```
#include "hpss_api.h"
unsigned32
hpss_HandleCompare(
    ns_ObjHandle_t    *FirstObjHandle,    /* IN */
    ns_ObjHandle_t    *SecondObjHandle); /* IN */
```

### Description

This function compares some fields in two object handles to determine if the handles map to the same object. True (1) is returned if the handles are for the same object and false (0) is returned if they are not for the same object.

### Parameters

<i>FirstObjHandle</i>	Handle to the first object for comparison.
<i>SecondObjHandle</i>	Handle to the second object for comparison.

### Return Values

Upon successful completion, a value of zero (false) or one (true) is returned . Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EINVAL	One or both object handles point to an invalid object type or are NULL pointers. Valid object types are: NS_OBJECT_TYPE_DIRECTORY NS_OBJECT_TYPE_JUNCTION NS_OBJECT_TYPE_FILE NS_OBJECT_TYPE_HARD_LINK NS_OBJECT_TYPE_SYM_LINK
--------	--

#### See Also

**hpss\_GetObjType, hpss\_GetObjId**

#### Notes

None.

## 2.1.75. hpss\_InsertDistFile

#### Purpose

Insert a distributed file given the file's information.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_InsertDistFile(
    api_dist_file_info_t    *FileInfo); /* IN */
```

#### Description

The **hpss\_InsertDistFile** function attempts to insert an extracted file table entry into the current file table.

#### Parameters

<i>FileInfo</i>	Pointer to file table information.
-----------------	------------------------------------

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value.

#### Error Conditions

EFAULT	<i>FileInfo</i> is a NULL pointer.
EINVAL	Checksum on file information failed.
EMFILE	Too many open file descriptors.

#### See Also

**hpss\_GetDistFile**

#### Notes

1. A file entry can only be inserted if it had previously been extracted using **hpss\_GetDistFile**.
2. The process will be killed if the file descriptor table is inconsistent.

## 2.1.76. hpss\_JunctionCreate

### Purpose

Create a junction to an HPSS fileset or directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_JunctionCreate(
    char          *Path,          /* IN */
    ns_ObjHandle_t *SourceHandle, /* IN */
    ns_ObjHandle_t *JunctionHandle ); /* OUT */
```

### Description

The **hpss\_JunctionCreate** function is called to create a HPSS junction to the specified directory or fileset handle.

### Parameters

<i>Path</i>	Specifies path name of the new junction.
<i>SourceHandle</i>	Points to the directory or fileset handle that is used for the source of the new junction.
<i>JunctionHandle</i>	Specifies the returned handle for the newly created junction.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value.

### Error Conditions

EFAULT	Either the <i>Path</i> , <i>SourceHandle</i> , or <i>JunctionHandle</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system imposed limit, or a component of the pathname exceeds the system imposed limit.
ENOENT	The <i>Path</i> argument points to an empty string.
EEXIST	The named path already exists in the HPSS name space.
EACCES	The requesting client is not the root user or a trusted user with write permissions.
EINVAL	The <i>SourceHandle</i> parameter doesn't point to a directory handle.

### See Also

**hpss\_FilesetCreate**, **hpss\_GetJunctions**, **hpss\_JunctionCreateHandle**, **hpss\_JunctionDelete**, **hpss\_JunctionDeleteHandle**.

### Notes

None.

## 2.1.77. hpss\_JunctionCreateHandle

### Purpose

Create a junction to an HPSS fileset or directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_JunctionCreateHandle(
    ns_ObjHandle_t    *ParentHandle,          /* IN */
    char              *Path,                  /* IN */
    ns_ObjHandle_t    *SourceHandle,          /* IN */
    sec_cred_t        *Ucred,                 /* IN */
    ns_ObjHandle_t    *JunctionHandle );      /* OUT */
```

### Description

The **hpss\_JunctionCreateHandle** function is called to create a HPSS junction to the specified directory or fileset handle.

### Parameters

<i>ParentHandle</i>	Specifies the directory for the new junction.
<i>Path</i>	Specifies path name of the new junction.
<i>SourceHandle</i>	Points to the directory or fileset handle that is used for the source of the new junction.
<i>Ucred</i>	User credentials.
<i>unctionHandle</i>	<i>Specifies the returned handle for the newly created junction.</i>

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value.

### Error Conditions

EFAULT	Either the <i>Path</i> , <i>SourceHandle</i> , or <i>JunctionHandle</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system imposed limit, or a component of the pathname exceeds the system imposed limit.
ENOENT	The <i>Path</i> argument points to an empty string.
EEXIST	The named path already exists in the HPSS name space.
EACCES	The requesting client is not the root user or a trusted user with write permissions.
EINVAL	The <i>SourceHandle</i> parameter doesn't point to a directory handle or the <i>Path</i> is invalid.

### See Also

**hpss\_FilesetCreate, hpss\_GetJunctions, hpss\_JunctionCreate, hpss\_JunctionDelete, hpss\_JunctionDeleteHandle.**

#### Notes

None.

## 2.1.78. hpss\_JunctionDelete

#### Purpose

Delete a junction.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_JunctionDelete(
    char      *Path );    /* IN */
```

#### Description

The **hpss\_JunctionDelete** is called to delete the junction specified by the *Path* input parameter.

#### Parameters

*Path* Specifies the name of the junction to be deleted.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value.

#### Error Conditions

ENOENT	The <i>Path</i> parameter is an empty string or doesn't refer to an existing object
EFAULT	The <i>Path</i> parameter is NULL.
EINVAL	The <i>Path</i> parameter doesn't refer to a junction.
EACCES	The requesting client is not the root user or a trusted user with write permissions.

#### See Also

**hpss\_JunctionCreate, hpss\_JunctionCreateHandle, hpss\_JunctionDeleteHandle.**

#### Notes

None.

## 2.1.79. hpss\_JunctionDeleteHandle

#### Purpose

Delete a junction.

#### Synopsis

```
#include "hpss_api.h"
```

```

int
hpss_JunctionDeleteHandle(
    ns_ObjHandle_t    *ParentHandle,    /* IN */
    char              *Path,            /* IN */
    sec_cred_t        *Ucred) ;        /* IN */

```

### Description

The **hpss\_JunctionDeleteHandle** is called to delete the junction specified by the *ParentHandle* and *Path* input parameters.

### Parameters

<i>ParentHandle</i>	Parent directory.
<i>Path</i>	Specifies the name of the junction to be deleted.
<i>Ucred</i>	User credentials.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value.

### Error Conditions

ENOENT	The <i>Path</i> parameter is an empty string or doesn't refer to an existing object
EFAULT	The <i>Path</i> parameter is NULL.
EINVAL	The <i>Path</i> parameter doesn't refer to a junction or the <i>ParentHandle</i> is NULL.
EACCES	The requesting client is not the root user or a trusted user with write permissions.

### See Also

**hpss\_JunctionCreate**, **hpss\_JunctionCreateHandle**, **hpss\_JunctionDelete**.

### Notes

If *Ucred* is NULL, the credentials in the current thread context are used.

## 2.1.80. hpss\_Link

### Purpose

Create a hard link to an existing HPSS file.

### Synopsis

```

#include "hpss_api.h"
int
hpss_Link(
    char    *Existing,    /* IN */
    char    *New );      /* IN */

```

### Description

The **hpss\_Link** routine creates a hard link to an existing file (hard links to directories are not currently



supported), given the path name of the existing file, *Existing*, and the path name of the new link, *New*.

### Parameters

<i>Existing</i>	Specifies the path name of the existing file to which the link is to be created.
<i>New</i>	Specifies the path name of the new link.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.
EEXIST	The object identified by <i>New</i> already exists.
EFAULT	The <i>Existing</i> or <i>New</i> parameter is a NULL pointer.
EPERM	The object specified by <i>Existing</i> is a directory.
EMLINK	The object specified by <i>Existing</i> is a directory.
ENAMETOOLONG	The length of the <i>Existing</i> or <i>New</i> argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	No entry exists for the specified file.
ENOTDIR	A component of the path prefix is not a directory.

### See Also

**hpss\_LinkHandle**, **hpss\_Symlink**, **hpss\_Unlink**, **hpss\_UnlinkHandle**.

### Notes

None.

## 2.1.81. hpss\_LinkHandle

### Purpose

Create a hard link to an existing HPSS file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_LinkHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    ns_ObjHandle_t    *DirHandle,    /* IN */
    char               *New,          /* IN */
    sec_cred_t         *Ucred) ;     /* IN */
```

### Description

The **hpss\_LinkHandle** function creates a new link to the file associated with *ObjHandle*, with the name *New*, taken relative to the directory specified by *DirHandle*.

## Parameters

<i>ObjHandle</i>	Specifies the handle of the existing file to which the link is to be created.
<i>DirHandle</i>	Handle to the parent directory.
<i>New</i>	Specifies the path name of the new link.
<i>Ucred</i>	User credentials.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the new link.
EEXIST	The object identified by <i>New</i> already exists.
EFAULT	The <i>ObjHandle</i> , <i>DirHandle</i> , or <i>New</i> parameter is a NULL pointer.
EPERM	The object specified by <i>ObjHandle</i> is a directory.
EMLINK	The number of links to the file named by <i>ObjHandle</i> would exceed the system-imposed limit.
ENAMETOOLONG	The length of the <i>New</i> argument exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	<i>New</i> points to a null string.
ENOTDIR	A component of the path prefix is not a directory.

## See Also

**hpss\_Link, hpss\_Symlink, hpss\_Unlink, hpss\_UnlinkHandle.**

## Notes

None.

## 2.1.82. hpss\_LoadThreadState

### Purpose

Updates the user credentials and file or directory creation mask for the current thread's Client API state.

### Synopsis

```
#include "hpss_api.h"
int
hpss_LoadThreadState(
    uid_t      UserID,          /* IN */
    mode_t     Umask,          /* IN */
    char       *ClientFullName ); /* IN */
```

### Description

The `hpss_LoadThreadState` routine updates the user credentials and file or directory creation mask found in the current thread's Client API state.

#### Parameters

<i>UserID</i>	Specifies the user ID for the user whose credentials are to be loaded.
<i>Umask</i>	Specifies the new file or directory creation mask.
<i>ClientFullName</i>	Specifies the client's fully qualified name.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

ENOENT	Credentials for the specified user could not be obtained.
--------	---

#### See Also

**hpss\_LoadDefaultThreadState**

#### Notes

None.

## 2.1.83. hpss\_LoadDefaultThreadState

#### Purpose

Special interface to allow well-behaved clients to manipulate the global thread state so that all threads will use the new state.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_LoadDefaultThreadState(
    uid_t      UserID,          /* IN */
    mode_t     Umask,          /* IN */
    char       *ClientFullName ); /* IN */
```

#### Description

The **hpss\_LoadDefaultThreadState** routine updates the global thread state so that all threads will use the new state. After this call, **hpss\_LoadThreadState** routine is effectively disabled because for each new API the credentials are reloaded from the global thread state.

#### Parameters

<i>UserID</i>	Specifies the user ID for the user whose credentials are to be loaded.
<i>Umask</i>	Specifies the new file or directory creation mask.
<i>ClientFullName</i>	Specifies the client fully qualified name.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

ENOENT                      Credentials for the specified user could not be obtained.

#### See Also

**hpss\_LoadThreadState**

#### Notes

None.

## 2.1.84. hpss\_Lseek

#### Purpose

Set the current file offset for an open file, given a 32-bit value.

#### Synopsis

```
#include <unistd.h>
#include "hpss_api.h"
hpss_off_t
hpss_Lseek(
    int          Fildes,      /* IN */
    hpss_off_t   Offset,      /* IN */
    int          Whence );    /* IN */
```

#### Description

The **hpss\_Lseek** function sets the file offset for the open file handle, *Fildes*. Refer to POSIX.1 for more detailed information.

#### Parameters

<i>Fildes</i>	Specifies the open file handle for which the file offset is to be set.
<i>Offset</i>	Specifies the number of bytes to be used in calculating the new file offset - dependent on the value of <i>Whence</i> as to the final effect on the new file offset.
<i>Whence</i>	Specifies how to interpret the <i>Offset</i> parameter in setting the file pointer associated with the <i>Fildes</i> parameter. Values for the <i>Whence</i> parameter are as follows:  SEEK_SET - file offset set to <i>Offset</i> .  SEEK_CUR - file offset set to current offset plus <i>Offset</i> .  SEEK_END - file offset set to current end of file plus <i>Offset</i> .

#### Return Values

Upon successful completion, **hpss\_Lseek** returns a nonnegative value representing the resulting offset as measured in bytes from the beginning the file. Otherwise, **hpss\_Lseek** returns a negative value; the absolute value of which is equal to an errno value set by POSIX.1 **lseek**.

## Error Conditions

EBADF	The specified file descriptor does not refer to an open file.
EBUSY	The file is currently in use by another client thread.
EFBIG	Could not represent the resulting offset in the return value.
EINVAL	The <i>Whence</i> parameter is invalid or the resulting offset would be invalid.
ESTALE	Connection to this file descriptor is no longer valid.

## See Also

**hpss\_Read**, **hpss\_Write**, **hpss\_SetFileOffset**.

## Notes

None.

## 2.1.85. hpss\_Lstat

### Purpose

Get file status (POSIX), returning status about a symbolic link if the named file is a symbolic link.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Lstat(
    char          *Path,      /* IN */
    hpss_stat_t   *Buf );    /* OUT */
```

### Description

The **hpss\_Lstat** function obtains information about the file named by *Path* and returns it in the structure pointed to by *Buf*. Refer to POSIX.1 for more detailed information. This function differs from **hpss\_Stat**, however, in that if the named file is a symbolic link, information is returned about the link itself, not about the file to which the link points. If *Path* refers to a junction, the junction will be traversed.

### Parameters

<i>Path</i>	Points to the path name of the file being queried.
<i>Buf</i>	Points to an <i>hpss_stat_t</i> structure that will contain the information for the file.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an *errno* value set by POSIX.1 **stat**.

## Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>Buf</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**, **hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileSetAttributes**, **hpss\_FileSetAttributesHandle**, **hpss\_FilesetAttributesSOID**, **hpss\_Stat**, **hpss\_Fstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

#### Notes

None.

## 2.1.86. hpss\_Migrate

#### Purpose

Migrate a file from a specified level in the storage hierarchy.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Migrate(
    int          Fildes,          /* IN */
    unsigned32   SrcLevel,        /* IN */
    unsigned32   Flags,           /* IN */
    u_signed64   *RetBytesMigrated ); /* OUT */
```

#### Description

The **hpss\_Migrate** routine migrates an open file from a level in the storage hierarchy, specified by *SrcLevel*. The *Flags* argument is used to control behavior of the request.

#### Parameters

<i>Fildes</i>	Specifies the file descriptor corresponding to the file to be migrated.
<i>SrcLevel</i>	Identifies the level in the storage hierarchy from which the data is to be migrated.
<i>Flags</i>	Controls the behavior of the migrate request. Anticipated values include: HPSS_MIGRATE_FORCE - migrate data even with copies at lower levels. HPSS_MIGRATE_PURGE_DATA - purge data after migration. HPSS_MIGRATE_NO_COPY - do not migrate multiple copies.
<i>RetBytesMigrated</i>	Points to an area to contain the number of bytes migrated.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EBADF	The supplied file descriptor does not correspond to a file opened for writing.
-------	--

EBUSY	The specified file descriptor is in use.
EFAULT	The <i>RetBytesMigrated</i> parameter is a NULL pointer.
EINVAL	The <i>Flags</i> argument is invalid.
EPERM	The client does not have the appropriate privileges to issue explicit file migration requests.

#### See Also

**hpss\_Purge**, **hpss\_Stage**.

#### Notes

None.

## 2.1.87. hpss\_Mkdir

#### Purpose

Create a directory.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Mkdir(
    char      *Path,      /* IN */
    mode_t    Mode );    /* IN */
```

#### Description

The **hpss\_Mkdir** function creates a new directory with the name *Path*. The file permission bits of the new directory are initialized by *Mode* and modified by the file creation mask of the thread.

#### Parameters

<i>Path</i>	Specifies the path name to be used for the newly created directory.
<i>Mode</i>	Specifies permission bits to be used in setting the mode of the new directory.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **mkdir**.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
EEXIST	The named file exists.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EMLINK	The link count of the parent directory would exceed the maximum allowed number of links.

ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_MkdirHandle**, **hpss\_Umask**, **hpss\_Rmdir**.

#### Notes

None.

## 2.1.88. hpss\_MkdirHandle

### Purpose

Create a directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_MkdirHandle(
    ns_ObjHandle_t    *ObjHandle,      /* IN */
    char              *Path,           /* IN */
    mode_t             Mode,           /* IN */
    sec_cred_t         *Ucred,         /* IN */
    ns_ObjHandle_t     *HandleOut,     /* OUT */
    hpss_vattr_t       *AttrOut);      /* OUT */
```

### Description

The **hpss\_MkdirHandle** function creates a new directory with the name *Path*, taken relative to the directory indicated by *ObjHandle*. The directory permission bits of the new directory are initialized by *Mode*, and modified by the file creation mask of the thread. The newly created directory's object handle and attributes are returned in the areas pointed to by *HandleOut* and *AttrOut* respectively.

### Parameters

<i>ObjHandle</i>	Handle of parent directory.
<i>Path</i>	Specifies the path name to be used for the newly created directory.
<i>Mode</i>	Specifies permission bits to be used in setting the mode of the new directory.
<i>Ucred</i>	User credentials. If NULL, credentials in current thread context are used.
<i>HandleOut</i>	New directory's object handle.
<i>AttrOut</i>	Returned VFS attributes.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **mkdir**.

### Error Conditions



EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.
EEXIST	The named file exists.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EMLINK	The link count of the parent directory would exceed the maximum allowed number of links.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_Mkdir, hpss\_Umask, hpss\_Rmdir, hpss\_RmdirHandle**

#### Notes

None.

## 2.1.89. hpss\_Open

#### Purpose

Optionally create and open an HPSS file.

#### Synopsis

```
#include <fcntl.h>
#include "hpss_api.h"
int
hpss_Open (
    char                *Path,           /* IN */
    int                 Oflag,          /* IN */
    mode_t              Mode,           /* IN */
    hpss_cos_hints_t    *HintsIn,       /* IN */
    hpss_cos_priorities_t *HintsPri,    /* IN */
    hpss_cos_hints_t    *HintsOut );    /* OUT */
```

#### Description

The **hpss\_Open** function establishes the connection between a file, named by the *Path* argument, and a file handle. If **O\_CREAT** is specified in *Oflag* and the file does not exist, an attempt will be made to create the file.

#### Parameters

*Path*                      Names the file to be opened or created.

<i>Oflag</i>	Specifies the file status and file access modes to be assigned. Applicable values given below may be OR'ed together. Refer to POSIX.1 for specific behavior. O_RDONLY O_WRONLY O_RDWR O_APPEND O_CREAT O_EXCL O_TRUNC O_NONBLOCK
<i>Mode</i>	Specifies the file mode for a file that is created as a result of O_CREAT.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used. This parameter is only used during file creation.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsPri</i> structure. This parameter is only used during file creation, and may be a NULL pointer. The priority will be set to REQUIRED_PRIORITY if <i>HintsPri</i> is NULL.
<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsPri</i> structure. This parameter is only used during file creation, and may be a NULL pointer. The priority will be set to REQUIRED_PRIORITY if <i>HintsPri</i> is NULL.
<i>HintsOut</i>	Points to an <b>hpss_cos_hints_t</b> structure which will contain the values actually used when the file is created. This argument may be a NULL pointer if the returned values are to be ignored. This parameter is only used during file creation.

## Return Values

Upon successful completion, **hpss\_Open** returns a nonnegative value that is the newly allocated file handle. Otherwise, **hpss\_Open** returns a negative value, the absolute value of which is equal to an errno value set by POSIX.1 **open**.

## Error Conditions

EACCES	One of the following conditions occurred: <ul style="list-style-type: none"> <li>● Search permission is denied on a component of the path prefix.</li> <li>● The file exists and the permissions specified by <i>Oflag</i> are denied.</li> <li>● The file doesn't exist and write permission is denied for the parent directory of the file to be created.</li> <li>● O_TRUNC is specified and write permission is denied.</li> </ul>
EEXIST	O_CREAT and O_EXCL are set and the named file exists.

EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
EINVAL	<i>Oflag</i> is not valid, or one or more values input in the <i>HintsIn</i> parameter is invalid.
EISDIR	The named file is a directory. Note that opening directories via <b>hpss_Open</b> is not supported in any mode.
EMFILE	The client open file table is already full.
ENFILE	Too many files are open in the system.
ENAMETOOLONG	The length of the <i>Path</i> string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	The named file does not exist and the O_CREAT flag was not specified, or the <i>Path</i> argument points to an empty string.
ENOMEM	Memory could not be allocated for the new path name.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_OpenHandle**, **hpss\_Close**, **hpss\_Umask**, **hpss\_OpenBitfile**, **hpss\_Create**, **hpss\_CreateHandle**, **hpss\_ReopenBitfile**.

#### Notes

Note that opening directories with **hpss\_Open** is not supported.

## 2.1.90. hpss\_OpenBitfile

#### Purpose

Open an HPSS bitfile, specified by bitfile ID.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_OpenBitfile(
    hpssoid_t          *BitFileID,      /* IN */
    int                OFlag,           /* IN */
    sec_cred_t         *Ucred,          /* IN */
    hpss_authz_token_t *AuthzTicket ); /* IN */
```

#### Description

The **hpss\_OpenBitfile** routine attempts to open the bitfile identified by *BitFileID*. Note that this routine cannot be used to create a bitfile; rather, **hpss\_Create**, **hpss\_CreateHandle**, **hpss\_CreateDMHandle**, **hpss\_Open** or **hpss\_OpenHandle** must be used for this purpose.

#### Parameters

*BitFileID* Points to bitfile identifier. The *BitfileID* is usually obtained using **hpss\_FileGetAttributes**.

<i>Oflags</i>	Specifies file status and file access modes to be assigned. Applicable values given below may be OR'ed together. Refer to POSIX.1 for specific behavior.
	O_RDONLY
	O_WRONLY
	O_RDWR
	O_APPEND
	O_TRUNC
<i>Ucred</i>	Points to client's user credentials.
<i>AuthzTicket</i>	Points to client's authorization for this file.

## Return Values

Upon successful completion, a nonnegative file descriptor is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	The client does not have permission for the requested file access.
EFAULT	The <i>BitFileID</i> or <i>AuthzTicket</i> parameter is a NULL pointer.
EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
EINVAL	<i>Oflag</i> is not valid.
EMFILE	The client file table is already full.
ENFILE	Too many bitfiles are already open in the system.
ENOENT	No entry exists for the specified bitfile ID.

## See Also

**hpss\_Create, hpss\_CreateHandle, hpss\_CreateDMHandle, hpss\_Open, hpss\_OpenHandle, hpss\_OpenBitfileVAttr, hpss\_ReopenBitfile, hpss\_Close**

## Notes

The user cannot generate a valid Authorization Ticket. Authorization Tickets are typically used by authorized clients. Unauthorized clients should pass a zero value for *AuthzTicket*.

## 2.1.91. hpss\_OpenBitfileVAttr

### Purpose

Open an HPSS bitfile, specified by its file attributes structure.

### Synopsis

```
#include "hpss_api.h"
int
hpss_OpenBitfileVAttr(
    hpss_vattr_t      *FileAttrs,      /* IN */
    int               Oflag,           /* IN */
    ...)
```

```

sec_cred_t          *Ucred,          /* IN */
hpss_authz_token_t  *AuthzTicket,    /* IN */
hpss_cos_hints_t    *HintsOut);      /* OUT */

```

## Description

The **hpss\_OpenBitfileVAttr** function is a simple interface for opening a bitfile using existing bitfile attributes and authorization ticket. This function is different from other open calls in that it only obtains an open context, without making any extra path traverse or get attribute calls.

## Parameters

<i>FileAttrs</i>	Points to the file's attributes structure.
<i>Oflags</i>	Specifies file status and file access modes to be assigned. Applicable values given below may be OR'ed together. Refer to POSIX.1 for specific behavior. O_RDONLY O_WRONLY O_RDWR O_APPEND O_TRUNC
<i>Ucred</i>	Points to client's user credentials.
<i>AuthzTicket</i>	Points to client's authorization for this file.
<i>HintsOut</i>	Pointer to hints used for opening file.

## Return Values

Upon successful completion, a nonnegative file descriptor is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	The client does not have permission for the requested file access.
EFAULT	The <i>BitFileID</i> or <i>AuthzTicket</i> parameter is a NULL pointer.
EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
EINVAL	<i>Oflag</i> is not valid.
EMFILE	The client file table is already full.
ENFILE	Too many bitfiles are already open in the system.
ENOENT	No entry exists for the specified bitfile ID.

## See Also

**hpss\_Create, hpss\_CreateHandle, hpss\_CreateDMHandle, hpss\_Open, hpss\_OpenHandle, hpss\_OpenBitfile, hpss\_ReopenBitfile, hpss\_Close**

## Notes

The user cannot generate a valid Authorization Ticket. Authorization Tickets are typically used by authorized clients. Unauthorized clients should pass a zero value for *AuthzTicket*.

## 2.1.92. hpss\_OpenHandle

### Purpose

Optionally create and open an HPSS file relative to a given directory.

### Synopsis

```
#include <fcntl.h>
#include "hpss_api.h"
int
hpss_OpenHandle(
    ns_ObjHandle_t      *ObjHandle,      /* IN */
    char                *Path,           /* IN */
    int                 Oflag,           /* IN */
    mode_t              Mode,            /* IN */
    sec_cred_t          *Ucred,          /* IN */
    hpss_cos_hints_t    *HintsIn,        /* IN */
    hpss_cos_priorities_t *HintsPri,     /* IN */
    hpss_cos_hints_t    *HintsOut,       /* OUT */
    hpss_vattr_t         *AttrsOut,      /* OUT */
    hpss_authz_token_t  *AuthzTicket);   /* OUT */
```

### Description

The **hpss\_OpenHandle** function establishes the connection between a file, named by the *Path* argument taken relative to the directory specified by *ObjHandle*, and a file handle. If *O\_CREAT* is specified in *Oflag* and the file does not exist, an attempt will be made to create the file.

### Parameters

<i>ObjHandle</i>	Parent object handle, the parent directory.
<i>Path</i>	Names the file to be opened or created.
<i>Oflags</i>	Specifies the file status and file access modes to be assigned. Applicable values given below may be OR'ed together. Refer to POSIX.1 for specific behavior. O_RDONLY O_WRONLY O_RDWR O_APPEND O_CREAT O_EXCL O_TRUNC
<i>Mode</i>	Specifies the file mode for a file that is created as a result of <i>O_CREAT</i> .
<i>Ucred</i>	Points to client's user credentials.
<i>HintsIn</i>	Points to an <b>hpss_cos_hints_t</b> structure which provides allocation hints to HPSS as to the expected structure or access of the file. This argument may be a NULL pointer if the default COS is to be used. This parameter is only used during file creation.

<i>HintsPri</i>	Points to an <b>hpss_cos_priorities_t</b> structure which provides the relative priorities associated with the fields contained in the <i>HintsPri</i> structure. This parameter is only used during file creation, and may be a NULL pointer. The priority will be set to REQUIRED_PRIORITY if <i>HintsPri</i> is NULL.
<i>HintsOut</i>	Points to an <b>hpss_cos_hints_t</b> structure which will contain the values actually used when the file is created. This argument may be a NULL pointer if the returned values are to be ignored. This parameter is only used during file creation.
<i>AttrsOut</i>	Returned file attributes.
<i>AuthzTicket</i>	Client Authorization.

## Return Values

Upon successful completion, **hpss\_OpenHandle** returns a nonnegative value that is the newly allocated file handle. Otherwise, **hpss\_OpenHandle** returns a negative value; the absolute value of which is equal to an errno value set by POSIX.1 **open**.

## Error Conditions

EACCES	One of the following conditions occurred: <ul style="list-style-type: none"> <li>● Search permission is denied on a component of the path prefix.</li> <li>● The file exists and the permissions specified by <i>Oflag</i> are denied.</li> <li>● The file doesn't exist and write permission is denied for the parent directory of the file to be created.</li> <li>● O_TRUNC is specified and write permission is denied.</li> </ul>
EEXIST	O_CREAT and O_EXCL are set and the named file exists.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
EINVAL	<i>Oflag</i> is not valid, or one or more values input in the <i>HintsIn</i> parameter is invalid.
EISDIR	The named file is a directory. Note that opening directories via <b>hpss_Open</b> is not supported in any mode.
EMFILE	The client open file table is already full.
ENFILE	Too many files are open in the system.
ENAMETOOLONG	The length of the <i>Path</i> string exceeds the system-imposed path name limit or a path name component exceeds the system-imposed limit.
ENOENT	The named file does not exist and the O_CREAT flag was not specified, or the <i>Path</i> argument points to an empty string.
ENOSPC	Resources could not be allocated for the new file.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Open**, **hpss\_Close**, **hpss\_Umask**, **hpss\_OpenBitfile**, **hpss\_Create**, **hpss\_CreateHandle**,

**hpss\_ReopenBitfile.**

#### Notes

Opening directories with **hpss\_OpenHandle** is not supported. Use **hpss\_Opendir** instead.

## 2.1.93. hpss\_Opendir

#### Purpose

Open an HPSS directory.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Opendir(
    char          *DirName );      /* IN */
```

#### Description

The **hpss\_Opendir** function opens a directory stream corresponding to the directory named by *DirName*. The directory stream is positioned at the first entry in the directory.

#### Parameters

*DirName*                      Specifies the path name of the directory to be opened.

#### Return Values

Upon successful completion, **hpss\_Opendir** returns a nonnegative value that is the newly allocated directory stream handle. Otherwise, **hpss\_Opendir** returns a negative value; the absolute value of which is equal to an errno value set by POSIX.1 **opendir**.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the directory itself.
EFAULT	The <i>DirName</i> parameter is a NULL pointer.
EMFILE	The open file table is already full.
ENAMETOOLONG	The length of the <i>DirName</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>DirName</i> argument points to an empty string.
ENOTDIR	A component of <i>DirName</i> is not a directory.

#### See Also

**hpss\_Readdir**, **hpss\_Rewinddir**, **hpss\_Closedir**.

#### Notes

The return value is changed from POSIX, primarily to make handling open directories and files in the client API consistent.



## 2.1.94. hpss\_OpenWithHints

### Purpose

Open an HPSS file using hints for COS and priorities.

### Synopsis

```
#include "hpss_posix_api.h"
int
hpss_OpenWithHints(
    char            *Path,            /* IN */
    int             Oflag,           /* IN */
    mode_t          Mode,            /* IN */
    hpss_cos_hints_t *HintsIn,       /* IN */
    hpss_cos_priorities_t *HintsPri, /* IN */
    hpss_cos_hints_t *HintsOut);    /* OUT */
```

### Description

The **hpss\_OpenWithHints** routine opens an HPSS file using input hints to define the COS and priorities.

### Parameters

<i>Path</i>	Specifies the path/name of the HPSS file.
<i>Oflag</i>	Specifies the file access. O_RDONLY O_WRONLY O_RDWR O_APPEND O_CREAT O_EXCL O_TRUNC
<i>Mode</i>	If O_CREAT is specified and a file is created, this argument gives the mode used for determining the file mode for the created file.
<i>HintsIn</i>	Pointer to structure defining client preferences for class of service.
<i>HintsPri</i>	Pointer to structure defining the priorities of each COS structure field.
<i>HintsOut</i>	Pointer to structure defining COS with which file was created.

### Return Values

Upon successful completion **hpss\_OpenWithHints** returns a non-negative value representing the open file descriptor. If an error is encountered, a negative value will be returned whose absolute value indicates the error condition.

### Error Conditions

EFAULT	Null <i>Path</i> pointer.
ENOENT	<i>Path</i> points to null string.

ENOMEM	Memory allocation failure.
EINVAL	Invalid hints parameters.

#### See Also

**hpss\_Open, hpss\_PosixOpen, hpss\_PosixCreate, hpss\_CreateWithHints, hpss\_PosixLseek**

#### Notes

The **hpss\_OpenWithHints** function is identical to **hpss\_Open**. It is created at the request of a customer.

## 2.1.95. hpss\_PIOEnd

#### Purpose

End a parallel I/O transfer.

#### Synopsis

```
int
hpss_PIOEnd(
    hpss_pio_grp_t    StripeGroup)    /* IN */
```

#### Description

This function verifies that the StripeGroup is valid and completes the I/O for the group if called by the coordinating side.

#### Parameters

*StripeGroup*                      Identifies the group of participants involved in the stripe group.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EINVAL	Invalid I/O stripe group input.
ENOMEM	Out of memory for stack entry.

#### See Also

**hpss\_PIOStart, hpss\_PIORegister, hpss\_PIOExecute, hpss\_PIOExportGrp, hpss\_PIOImportGrp**

#### Notes

None.

## 2.1.96. hpss\_PIOExecute

#### Purpose

Begin a parallel I/O transfer.

#### Synopsis

```
int
hpss_PIOExecute(
    int                Fd,                /* IN */
```

```

    u_signed64      FileOffset,    /* IN */
    u_signed64      Size,          /* IN */
    hpss_pio_grp_t  StripeGroup,   /* IN */
    hpss_pio_gapinfo_t *GapInfo,   /* OUT */
    u_signed64      *BytesMoved)    /* OUT */

```

### Description

This function is used by a transfer coordinator to begin a parallel I/O data transfer operation.

### Parameters

<i>Fd</i>	An open HPSS File descriptor.
<i>FileOffset</i>	The file offset.
<i>Size</i>	Size of the I/O.
<i>StripeGroup</i>	The stripe group.
<i>GapInfo</i>	Pointer to information concerning a sparse file.
<i>BytesMoved</i>	Number of bytes moved with this transfer.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EINVAL	Invalid argument.
ENOMEM	Not enough memory for stack entry.

### See Also

**hpss\_PIOStart, hpss\_PIORegister, hpss\_PIOEnd, hpss\_PIOExportGrp, hpss\_PIOImportGrp**

### Notes

None.

## 2.1.97. hpss\_PIOExportGrp

### Purpose

Convert group I/O data to raw data.

### Synopsis

```

int
hpss_PIOExportGrp(
    hpss_pio_grp_t  StripeGroup, /* IN */
    void            **Buffer,     /* OUT */
    unsigned int     *BufLength)  /* OUT */

```

### Description

This function is used by a transfer coordinator to convert parallel IO group context data into raw data that is suitable to transfer across the network.

### Parameters

<i>StripeGroup</i>	The I/O stripe group.
<i>Buffer</i>	Pointer to the output raw data buffer pointer.
<i>BufLength</i>	Pointer to the length of <i>Buffer</i> .

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT	<i>Buffer</i> or <i>BufLength</i> pointer is NULL.
EINVAL	Invalid <i>StripeGroup</i> input.
ENOMEM	Not enough memory to allocate output buffer.

### See Also

**hpss\_PIOEnd**, **hpss\_PIOStart**, **hpss\_PIORegister**, **hpss\_PIOImportGrp**, **hpss\_Execute**

### Notes

None.

## 2.1.98. hpss\_PIOImportGrp

### Purpose

Convert raw parallel I/O group context data.

### Synopsis

```
int
hpss_PIOImportGrp(
    void *Buffer,           /* IN */
    unsigned int BufLength, /* IN */
    hpss_pio_grp_t *StripeGroup) /* OUT */
```

### Description

This function converts raw parallel I/O group data into a format suitable for the **hpss\_PIORegister** function.

### Parameters

<i>Buffer</i>	Pointer to the input raw data buffer.
<i>BufLength</i>	Length of the raw data buffer.
<i>StripeGroup</i>	Pointer to the converted stripe group data.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EINVAL	Invalid parameter or bad checksum.
--------	------------------------------------

ENOMEM                      Not enough memory to allocate for the stripe group structure.

**See Also**

**hpss\_PIOExecute, hpss\_PIOEnd, hpss\_PIORegister, hpss\_PIOExportGrp, hpss\_PIOStart**

**Notes**

None.

## 2.1.99. hpss\_PIORegister

**Purpose**

Register an I/O callback function.

**Synopsis**

```
int
hpss_PIORegister(
    unsigned32      StripeElement, /* IN */
    unsigned32      DataNetAddr,   /* IN */
    void            *DataBuffer,   /* IN */
    unsigned32      DataBufLen,    /* IN */
    hpss_pio_grp_t  StripeGroup,   /* IN */
    hpss_pio_cb_t   IOCallback,    /* IN */
    void            *IOCallbackArg /* IN */)
```

**Description**

This function is used by a parallel I/O participant to register a callback function.

**Parameters**

<i>StripeElement</i>	Element of the data stripe.
<i>DataNetAddr</i>	Element of the data stripe.
<i>DataBuffer</i>	Buffer for the I/O.
<i>DataBufLen</i>	Length of the data buffer.
<i>StripeGroup</i>	The I/O stripe group.
<i>IOCallback</i>	I/O callback function.
<i>IOCallbackArg</i>	Pointer to the callback argument.

**Return Values**

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

**Error Conditions**

EINVAL	Invalid argument or checksum.
ENOMEM	Not enough memory to allocate stripe group.

**See Also**

**hpss\_PIOStart, hpss\_PIOEnd, hpss\_PIOExecute, hpss\_PIOImportGrp, hpss\_ExportGrp**

## 2.1.100. hpss\_PIOStart

### Purpose

Start a new parallel I/O group data transfer.

### Synopsis

```
int
hpss_PIOStart(
    hpss_pio_params_t *InputParams, /* IN */
    hpss_pio_grp_t    *StripeGroup) /* OUT */
```

### Description

This function is used by a transfer coordinator to start a new parallel I/O group context.

### Parameters

<i>InputParams</i>	Parameters defining the type of transport, stripe width, number of data elements
<i>StripeGroup</i>	Pointer to the stripe group.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT	<i>StripeGroup</i> is NULL.
EINVAL	Invalid or NULL <i>InputParams</i> .
ENOMEM	Not enough memory to allocate stripe group.

### See Also

**hpss\_PIOExecute, hpss\_PIOEnd, hpss\_PIOExportGrp, hpss\_PIOImportGrp, hpss\_PIORegister**

### Notes

None.

## 2.1.101. hpss\_Purge

### Purpose

Purge a piece of a file from a specified level in the storage hierarchy.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Purge(
    int          Fildes,          /* IN */
    u_signed64   Offset,          /* IN */
    u_signed64   Length,          /* IN */
    unsigned32   StorageLevel,    /* IN */
    unsigned32   Flags,           /* IN */
    u_signed64   *RetBytesPurged ); /* OUT */
```

## Description

The **hpss\_Purge** routine purges part of an open file, specified by *Fildes*, *Offset* and *Length* from a level in the storage hierarchy, specified by *StorageLevel*. The *Flags* argument is used to control behavior of the request.

## Parameters

<i>Fildes</i>	Specifies the file descriptor corresponding to the file to be purged.
<i>Offset</i>	Specifies the offset of the start of the data to be purged. Currently must be 0.
<i>Length</i>	Specifies the length of the data to be purged. Currently must be 0.
<i>StorageLevel</i>	Identifies the level in the storage hierarchy from which the data is to be purged.
<i>Flags</i>	Controls the behavior of the purge request. Valid values include: BFS_PURGE_ALL - purge the entire file (required).
<i>RetBytesPurged</i>	Points to an area that will contain the number of bytes purged.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	The supplied file descriptor does not correspond to an open file.
EBUSY	The specified file descriptor is in use.
EFAULT	The <i>RetBytesPurged</i> parameter is a NULL pointer.
EINVAL	he <i>Flags</i> , <i>Offset</i> or <i>Length</i> argument is invalid.
EPERM	The client does not have the appropriate privileges to perform explicit purge operations.

## See Also

**hpss\_Migrate**, **hpss\_Stage**, **hpss\_PurgeLock**

## Notes

None.

## 2.1.102. hpss\_PurgeLock

### Purpose

Lock (or unlock) a file into the top level of its hierarchy.

### Synopsis

```
#include "hpss_api.h"
int
hpss_PurgeLock (
    int          Fildes,      /* IN */
    purgelock_flag_t Flag ); /* IN */
```

## Description

The **hpss\_PurgeLock** routine either locks a file in the top level of its hierarchy from being purged or removes an existing lock.

## Parameters

<i>Fildes</i>	Specifies the file descriptor corresponding to the file to be locked/unlocked.
<i>Flag</i>	Controls whether the request locks or unlocks the file. Possible values are: PURGE_LOCK - Purge lock the file  PURGE_UNLOCK - Purge unlock the file.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	The supplied file descriptor does not correspond to an open file.
EBUSY	The specified file descriptor is in use.
ESTALE	The connection for this entry is not valid.

## See Also

**hpss\_Purge.**

## Notes

None.

## 2.1.103. hpss\_PurgeLoginCred

### Purpose

Purge login credentials set by **hpss\_SetLoginCred**.

### Synopsis

```
#include "hpss_api.h"
void
hpss_PurgeLoginCred(void);
```

### Description

This routine purges the login credentials established by the routine **hpss\_SetLoginCred**.

### Parameters

None.

### Return Values

None.

### Error Conditions



None.

#### See Also

**hpss\_SetLoginCred.**

#### Notes

None.

## 2.1.104. hpss\_Read

#### Purpose

Read a contiguous section of an HPSS file, beginning at the current file offset, into a client buffer.

#### Synopsis

```
#include "hpss_api.h"
ssize_t
hpss_Read(
    int          Fildes,      /* IN */
    void         *Buf,        /* IN */
    size_t       Nbyte );    /* IN */
```

#### Description

The **hpss\_Read** function attempts to read *Nbyte* bytes from the file associated with the open file handle, *Fildes*, into the client buffer pointed to by *Buf*.

#### Parameters

<i>Fildes</i>	Specifies the open file handle associated with the file from which data is to be read.
<i>Buf</i>	Point to a buffer where the data is to be placed.
<i>Nbyte</i>	Specifies the number of bytes to be read.

#### Return Values

Upon successful completion, **hpss\_Read** returns a nonnegative value that is the number of bytes read, including any holes encountered. Otherwise **hpss\_Read** returns a negative value, the absolute value of which is equal to an errno value set by POSIX.1 **read**.

#### Error Conditions

EBADF	The specified file descriptor does not correspond to a file opened for reading.
EBUSY	The file is currently in use by another client thread.
EFAULT	The <i>Buf</i> parameter is NULL.
EIO	An input/output or HPSS internal error occurred.

#### See Also

**hpss\_Open**, **hpss\_OpenHandle**, **hpss\_OpenBitfile**, **hpss\_OpenBitfileVAttrs**, **hpss\_OpenBitfileHandle**, **hpss\_Write**, **hpss\_Lseek**, **hpss\_SetFileOffset**, **hpss\_ReadList**, **hpss\_WriteList**.

## Notes

None.

## 2.1.105. hpss\_ReadAttrs

### Purpose

Read directory entries and optionally return entry attributes.

### Synopsis

```
#include "hpss_api.h"
int
hpss_ReadAttrs (
    int                Dirdes,          /* IN */
    u_signed64         OffsetIn,        /* IN */
    unsigned32         BufferSize,       /* IN */
    unsigned32         GetAttributes,    /* IN */
    unsigned32         *End,            /* OUT */
    u_signed64         *OffsetOut,       /* OUT */
    ns_DirEntry_t      *DirentPtr );    /* OUT */
```

### Description

The **hpss\_ReadAttrs** routine returns a list of directory entries, which optionally includes file or directory attributes.

### Parameters

<i>Dirdes</i>	Specifies the open directory stream handle corresponding to the directory being read.
<i>OffsetIn</i>	Specifies the starting directory offset. If zero, the list will be from the beginning of the directory.
<i>BufferSize</i>	Specifies the size of the buffer pointed to by <i>DirentPtr</i> , in bytes. The maximum number of entries that fit in the buffer will be returned, until the end of the directory is reached.
<i>GetAttributes</i>	Indicates, if nonzero, that attributes will be returned with the directory entries.
<i>End</i>	Points to an area that will contain an indication of whether the last entry is returned in the current list.
<i>OffsetOut</i>	Points to an area that will contain the directory offset at the end of the returned list. This value can be supplied to a subsequent call to continue with the next directory entry.
<i>DirentPtr</i>	Points to a buffer to hold the returned list of directory entries and attributes.

### Return Values

Upon successful completion, the return value indicates the number of directory entries in the returned list. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EBADF	The specified directory descriptor does not refer to an open directory.
EBUSY	The directory is currently in use by another client thread.
EFAULT	The <i>DirentPtr</i> , <i>End</i> or <i>OffsetOut</i> parameter is a NULL pointer.
EINVAL	The <i>BufferSize</i> parameter is zero.

#### See Also

**hpss\_ReadAttrsHandle**, **hpss\_Opendir**, **hpss\_Closedir**.

#### Notes

Calling **hpss\_ReadAttrs** does not affect the directory offset or cached directory entries that are manipulated via **hpss\_Readaddr** and **hpss\_Rewinddir**.

## 2.1.106. hpss\_ReadAttrsHandle

#### Purpose

Read directory entries and optionally return entry attributes.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_ReadAttrsHandle(
    ns_ObjHandle_t    *ObjHandle,        /* IN */
    u_signed64         OffsetIn,          /* IN */
    sec_cred_t         *Ucred,           /* IN */
    unsigned32         BufferSize,        /* IN */
    unsigned32         GetAttributes,    /* IN */
    unsigned32         *End,             /* OUT */
    u_signed64         *OffsetOut,       /* OUT */
    ns_DirEntry_t      *DirentPtr );    /* OUT */
```

#### Description

The **hpss\_ReadAttrsHandle** routine returns a list of directory entries, which optionally includes file or directory attributes.

#### Parameters

<i>ObjHandle</i>	The directory object handle.
<i>OffsetIn</i>	Specifies the starting directory offset. If zero, the list will be from the beginning of the directory.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>BufferSize</i>	Specifies the size of the buffer pointed to by <i>DirentPtr</i> , in bytes. The maximum number of entries that fit in the buffer will be returned, until the end of the directory is reached.
<i>GetAttributes</i>	Indicates, if nonzero, that attributes will be returned with the directory entries.
<i>End</i>	Points to an area that will contain an indication of whether the last entry is

	returned in the current list.
<i>OffsetOut</i>	Points to an area that will contain the directory offset at the end of the returned list. This value can be supplied to a subsequent call to continue with the next directory entry.
<i>DirentPtr</i>	Points to a buffer to hold the returned list of directory entries and attributes.

## Return Values

Upon successful completion, the return value indicates the number of directory entries in the returned list. Otherwise, a negative value is returned, the absolute value of which is equal to an `errno` value defined below.

## Error Conditions

EBADF	The specified directory descriptor does not refer to an open directory.
EBUSY	The directory is currently in use by another client thread.
EFAULT	The <i>DirentPtr</i> , <i>End</i> or <i>OffsetOut</i> parameter is a NULL pointer.
EINVAL	The <i>ObjHandle</i> pointer is NULL or the <i>BufferSize</i> parameter is zero.

## See Also

**hpss\_ReadAttrs**, **hpss\_ReadRawAttrsHandle**, **hpss\_Opendir**, **hpss\_Closedir**.

## Notes

Calling **hpss\_ReadAttrsHandle** does not affect the directory offset or cached directory entries that are manipulated via **hpss\_Readdir**, **hpss\_ReaddirHandle**, and **hpss\_Rewinddir**.

## 2.1.107. hpss\_Readdir

### Purpose

Read a directory entry.

### Synopsis

```
#include <dirent.h>
#include "hpss_api.h"
int
hpss_Readdir(
    int                DirDes,          /* IN */
    hpss_dirent_t      *DirentPtr );   /* OUT */
```

### Description

The **hpss\_Readdir** function returns a structure, *DirentPtr*, representing the directory entry at the current position in the open directory stream. Reference the *hpss\_dirent\_t* structure definition for more detailed information.

### Parameters

<i>DirDes</i>	Specifies the open directory stream handle corresponding to the directory being read.
---------------	---

*DirentPtr* Points to a structure that will contain the directory entry information.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **readdir**.

### Error Conditions

EBADF	The specified directory descriptor does not refer to an open directory.
EBUSY	The directory is currently in use by another client thread.
EFAULT	The <i>DirentPtr</i> parameter is a NULL pointer.

### See Also

**hpss\_Openidir**, **hpss\_Rewindidir**, **hpss\_Closedir**.

### Notes

1. **hpss\_Readdir** is altered from POSIX.1 *readdir* to be more consistent with other HPSS calls. These differences are that **hpss\_Readdir** 1) accepts an integer directory stream handle (see **hpss\_Openidir**) and 2) moves the returned structure pointer to the argument list rather than the return value.
2. When the end of the directory is encountered, the *d\_name* field will be set to an empty string, and the *d\_namelen* field will be set to zero. These fields are in *DirentPtr* structure.

## 2.1.108. hpss\_ReaddirHandle

### Purpose

Read a directory entry.

### Synopsis

```
#include <dirent.h>
#include "hpss_api.h"
int
hpss_ReaddirHandle(
    ns_ObjHandle_t    *ObjHandle,      /* IN */
    u_signed64         OffsetIn,        /* IN */
    sec_cred_t         *Ucred,          /* IN */
    unsigned32         BufferSize,      /* IN */
    unsigned32         *End,            /* OUT */
    u_signed64         *OffsetOut,      /* OUT */
    hpss_dirent_t      *DirentPtr );   /* OUT */
```

### Description

The **hpss\_ReaddirHandle** function returns a structure, *DirentPtr*, representing the directory entry at the current position in the open directory stream. Reference the *hpss\_dirent\_t* structure definition for more detailed information. The function also returns the resulting directory position in *OffsetOut*, and an end of directory in *End*.

### Parameters

<i>ObjHandle</i>	Specifies the open directory object handle corresponding to the directory being read.
<i>OffsetIn</i>	Current directory position.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>BufferSize</i>	Size of buffer pointed to by <i>DirentPtr</i> .
<i>End</i>	End of directory indicator.
<i>OffsetOut</i>	Resulting directory position.
<i>DirentPtr</i>	<i>Points to a structure that will contain the directory entry information.</i>

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **readdir**.

### Error Conditions

EBADF	The specified directory descriptor does not refer to an open directory.
EBUSY	The directory is currently in use by another client thread.
EFAULT	One of the <i>End</i> , <i>OffsetOut</i> , or <i>DirentPtr</i> parameters is NULL.
EINVAL	<i>BufferSize</i> is zero or <i>ObjHandle</i> is NULL.

### See Also

**hpss\_Opendir**, **hpss\_Rewinddir**, **hpss\_Closedir**.

### Notes

1. **hpss\_Readdir** is altered from POSIX.1 *readdir* to be more consistent with other HPSS calls. These differences are that **hpss\_Readdir** 1) accepts an integer directory stream handle (see **hpss\_Opendir**) and 2) moves the returned structure pointer to the argument list rather than the return value.
2. When the end of the directory is encountered, the *d\_name* field will be set to an empty string, and the *d\_namelen* field will be set to zero. These fields are in *DirentPtr* structure.

## 2.1.109. hpss\_Readlink

### Purpose

Read the value of a symbolic link (i.e., the data stored in the symbolic link).

### Synopsis

```
#include "hpss_api.h"
int
hpss_Readlink(
    char        *Path,           /* IN */
    char        *Contents,       /* OUT */
    size_t      BufferSize );    /* IN */
```

### Description

The **hpss\_Readlink** routine returns the value of a symbolic link (not including any terminating null character) specified by *Path* into the buffer specified by *Contents*. The size of the buffer is specified by *BufferSize*.

## Parameters

<i>Path</i>	Specifies the name of the symbolic link to be read.
<i>Contents</i>	Points to buffer to contain the value of the symbolic link.
<i>BufferSize</i>	Specifies the size of the buffer pointed to by <i>Contents</i> .

## Return Values

Upon successful completion, the length of the symbolic link name is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the symbolic link.
EFAULT	The <i>Path</i> or <i>Contents</i> parameter is a NULL pointer.
EINVAL	The specified file is not a symbolic link.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The specified path name does not exist.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
ERANGE	The size of the <i>Contents</i> buffer is not big enough to contain the contents of the symbolic link or the value of the <i>BufferSize</i> parameter is zero.

## See Also

**hpss\_ReadlinkHandle, hpss\_Symlink, hpss\_SymlinkHandle**

## Notes

None.

# 2.1.110. hpss\_ReadlinkHandle

## Purpose

Read the value of a symbolic link (i.e., the data stored in the symbolic link).

## Synopsis

```
#include "hpss_api.h"
int
hpss_ReadlinkHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char              *Path,          /* IN */
    char              *Contents,      /* OUT */
    size_t            BufferSize );   /* IN */
sec_cred_t          *Ucred)         /* IN */
```

## Description

The **hpss\_ReadlinkHandle** routine returns the value of a symbolic link (not including any terminating null character) specified by *Path* into the buffer specified by *Contents*. The symbolic link is taken relative to *ObjHandle*. The size of the buffer is specified by *BufferSize*.

## Parameters

<i>ObjHandle</i>	Parent object handle.
<i>Path</i>	Specifies the name of the symbolic link to be read.
<i>Contents</i>	Points to buffer to contain the value of the symbolic link.
<i>BufferSize</i>	Specifies the size, in bytes, of the buffer pointed to by <i>Contents</i> .
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

## Return Values

Upon successful completion, the length of the symbolic link name is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or read permission is denied on the symbolic link.
EFAULT	The <i>ObjHandle</i> , <i>Path</i> or <i>Contents</i> parameter is a NULL pointer.
EINVAL	The specified file is not a symbolic link.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The specified path name does not exist.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
ERANGE	The size of the <i>Contents</i> buffer is not big enough to contain the contents of the symbolic link or the value of the <i>BufferSize</i> parameter is zero.

## See Also

**hpss\_Readlink**, **hpss\_Symlink**, **hpss\_SymlinkHandle**

## 2.1.111. hpss\_ReadRawAttrsHandle

### Purpose

Read directory entries and optionally return entry attributes.

### Synopsis

```
#include "hpss_api.h"
int
hpss_ReadRawAttrsHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    u_signed64        OffsetIn,      /* IN */
    ...)
```



```

sec_cred_t      *Ucred,           /* IN */
unsigned32      BufferSize,        /* IN */
unsigned32      GetAttributes,    /* IN */
unsigned32      *End,             /* OUT */
u_signed64      *OffsetOut,       /* OUT */
ns_DirEntry_t   *DirentPtr );    /* OUT */

```

## Description

The **hpss\_ReadRawAttrsHandle** routine returns a list of directory entries, which optionally includes file or directory attributes.

## Parameters

<i>ObjHandle</i>	The directory object handle.
<i>OffsetIn</i>	Specifies the starting directory offset. If zero, the list will be from the beginning of the directory.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>BufferSize</i>	<i>Specifies the size of the buffer pointed to by DirentPtr, in bytes. The maximum number of entries that fit in the buffer will be returned, until the end of the directory is reached.</i>
<i>GetAttributes</i>	<i>Indicates, if nonzero, that attributes will be returned with the directory entries.</i>
<i>End</i>	<i>Points to an area that will contain an indication of whether the last entry is returned in the current list.</i>
<i>OffsetOut</i>	<i>Points to an area that will contain the directory offset at the end of the returned list. This value can be supplied to a subsequent call to continue with the next directory entry.</i>
<i>DirentPtr</i>	<i>Points to a buffer to hold the returned list of directory entries and attributes.</i>

## Return Values

Upon successful completion, the return value indicates the number of directory entries in the returned list. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	The specified directory descriptor does not refer to an open directory.
EBUSY	The directory is currently in use by another client thread.
EFAULT	The <i>DirentPtr</i> , <i>End</i> or <i>OffsetOut</i> parameter is a NULL pointer.
EINVAL	The <i>ObjHandle</i> pointer is NULL or the <i>BufferSize</i> parameter is zero.

## See Also

**hpss\_ReadAttrs**, **hpss\_ReadAttrsHandle**, **hpss\_Opendir**, **hpss\_Closedir**.

## Notes

1. Calling **hpss\_ReadRawAttrsHandle** does not affect the directory offset or cached directory entries that

are manipulated via **hpss\_Readdir**, **hpss\_ReaddirHandle**, and **hpss\_Rewinddir**.

2. This routine will not follow HPSS junctions.

## 2.1.112. **hpss\_Rename**

### Purpose

Rename a file or directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Rename (
    char      *Old,      /* IN */
    char      *New );   /* IN */
```

### Description

The **hpss\_Rename** function changes the name of the file or directory currently named by *Old* to *New*.

### Parameters

<i>Old</i>	Specifies the path name that currently names the file or directory.
<i>New</i>	Specifies the path name to which the name is to be changed.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **rename**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or one of the directories containing <i>Old</i> or <i>New</i> denies write permission, or write permission is required and denied for a directory pointed to by the <i>Old</i> or <i>New</i> arguments.
EFAULT	The <i>Old</i> or <i>New</i> parameter is a NULL pointer.
EISDIR	The <i>New</i> argument points to a directory, and the <i>Old</i> argument points to a file that is not a directory.
EMLINK	The file named by <i>Old</i> is a directory, and the link count of the parent directory of <i>New</i> already contains the maximum allowed number of links.
ENAMETOOLONG	The length of the <i>Old</i> or <i>New</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Old</i> or <i>New</i> argument points to an empty string.
ENOTDIR	A component of the path prefix is not a directory.
ENOTEMPTY	The path named by <i>New</i> is a directory containing entries other than dot and dot-dot.

### See Also

**hpss\_RenameHandle, hpss\_Unlink.**

#### Notes

None.

## 2.1.113. hpss\_RenameHandle

#### Purpose

Rename a file or directory.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_RenameHandle(
    ns_ObjHandle_t      *OldHandle,      /* IN */
    char                *OldPath,        /* IN */
    ns_ObjHandle_t      *NewHandle,      /* IN */
    char                *NewPath,        /* IN */
    sec_cred_t          *Ucred);        /* IN */
```

#### Description

The **hpss\_RenameHandle** function changes the name of the file or directory currently named by *OldPath*, taken relative to the directory indicated by *OldHandle* to the new name *NewPath*, taken relative to the directory indicated by *NewHandle*.

#### Parameters

<i>OldHandle</i>	Specifies the current parent object handle.
<i>OldPath</i>	Specifies the current path name relative to <i>OldHandle</i> .
<i>NewHandle</i>	Specifies the new parent object handle.
<i>NewPath</i>	Specifies the new path name relative to <i>NewHandle</i> .
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **rename**.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or one of the directories containing <i>OldHandle</i> or <i>NewHandle</i> denies write permission, or write permission is required and denied for a directory pointed to by the <i>OldHandle</i> or <i>NewHandle</i> arguments.
EFAULT	<i>NewPath</i> is a NULL pointer.
EINVAL	The <i>NewHandle</i> or <i>OldHandle</i> parameter is NULL.

EISDIR	The <i>NewHandle</i> argument points to a directory, and the <i>OldHandle</i> argument points to a file that is not a directory.
EMLINK	The file named by <i>OldHandle</i> is a directory, and the link count of the parent directory of <i>NewHandle</i> already contains the maximum allowed number of links.
ENAMETOOLONG	The length of <i>OldPath</i> or <i>NewPath</i> exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>OldPath</i> or <i>NewPath</i> argument points to an empty string.
ENOTDIR	A component of the path prefix is not a directory.
ENOTEMPTY	The path named by <i>NewPath</i> is a directory containing entries other than dot and dot dot.

#### See Also

**hpss\_Rename, hpss\_Unlink.**

#### Notes

None.

## 2.1.114. hpss\_ReopenBitfile

#### Purpose

Given the bitfile ID, reopen an HPSS file using the same file table entry.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_ReopenBitfile(
    int                Fildes,           /* IN */
    hpssoid_t          *BitFileID,       /* IN */
    int                Oflag,           /* IN */
    sec_cred_t          *Ucred,         /* IN */
    hpss_authz_token_t *AuthzTicket );  /* IN */
```

#### Description

The **hpss\_ReopenBitfile** routine reopens the bitfile specified by *BitFileID*, using the file table entry currently associated with *Fildes*.

#### Parameters

<i>Fildes</i>	Specifies the file handle for the currently open file.
<i>BitFileID</i>	Points to the bitfile ID of the file to be reopened.
<i>Oflags</i>	Specifies the file status and file access modes to be assigned. Applicable values given below may be OR'ed together. Refer to POSIX.1 for specific behavior. O_RDONLY O_WRONLY

	O_RDONLY
	O_APPEND
	O_TRUNC
<i>Ucred</i>	Points to client's user credentials.
<i>AuthzTicket</i>	Points to area where authorization ticket is to be returned.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EBADF	<i>Fildes</i> does not refer to an open file.
EBUSY	The open file descriptor is in use by another thread.
EFAULT	The <i>BitFileID</i> or <i>AuthzTicket</i> parameter is a NULL pointer.
EINPROGRESS	The file is currently being staged. The open should be retried at a later time.
EINVAL	<i>Oflag</i> does not contain a valid access mode or <i>BitfileID</i> or <i>AuthzTicket</i> is NULL.
ENOENT	No entry exists for the specified bitfile ID.

## See Also

**hpss\_Open**, **hpss\_OpenBitfile**.

## Notes

If this routine fails, the file table entry identified by *Fildes* is not freed (it is marked as STALE), so that a subsequent effort can be made for this same file table entry.

## 2.1.115. hpss\_ResetSubSysStats

### Purpose

Reset Storage Subsystem statistics.

### Synopsis

```
#include "hpss_api.h"
int
hpss_ResetSubSysStats(
    unsigned32      SubsystemID,      /* IN */
    subsys_stats_t  *StatsOut );      /* OUT */
```

### Description

The **hpss\_ResetSubSysStats** routine resets the stage, migration, purge, and delete counts in the Storage Subsystem and sets the last reset time to the current time.

### Parameters

<i>SubsystemID</i>	Subsystem identifier.
--------------------	-----------------------

<i>StatsOut</i>	Points to an area that will contain the current Storage Subsystem statistics values.
-----------------	--

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EFAULT	The <i>StatsOut</i> parameter is a NULL pointer.
--------	--

### See Also

**hpss\_GetSubSysStats.**

### Notes

None.

## 2.1.116. hpss\_Rewinddir

### Purpose

Reset position of an open directory stream.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Rewinddir(
    int          Dirdes );          /* IN */
```

### Description

The **hpss\_Rewinddir** function resets the position of an open directory stream corresponding to *Dirdes* to the beginning of that directory.

### Parameters

<i>Dirdes</i>	Specifies the open directory stream handle for which the position is to be reset.
---------------	---

### Return Values

Upon successful completion, a value of zero is returned. If an error is encountered, a negative value is returned whose absolute value is described below.

### Error Conditions

EBADF	The specified directory descriptor does not correspond to an open directory or <i>Dirdes</i> is negative, or there are too many open file and directory descriptors.
EBUSY	Another client thread is currently using this directory descriptor.
ESTALE	Connection for this entry no longer valid.

### See Also

**hpss\_Opendir, hpss\_Readdir, hpss\_ReaddirHandle, hpss\_Closedir.**

## Notes

**hpss\_Rewinddir** is altered from POSIX to return the values described above (whereas the POSIX **rewinddir** function does not return a value). Providing a failure indication was thought to be more important than strict POSIX compatibility.

## 2.1.117. hpss\_Rmdir

### Purpose

Remove an HPSS directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Rmdir(
    char      *Path );    /* IN */
```

### Description

The **hpss\_Rmdir** function removes the directory named by *Path*. The directory will only be removed if the directory is empty.

### Parameters

*Path*                      Specifies the path name of the directory to be removed.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **rmdir**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
EBUSY	The named directory is currently in use and cannot be removed.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

### See Also

**hpss\_RmdirHandle**, **hpss\_RmdirNoLookup**, **hpss\_Mkdir**, **hpss\_Unlink**.

### Notes

None.

## 2.1.118. hpss\_RmdirHandle

### Purpose

Remove an HPSS directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_RmdirHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char               *Path,        /* IN */
    sec_cred_t         *Ucred);      /* IN */
```

### Description

The **hpss\_RmdirHandle** function removes the directory named by *Path* taken relative to the directory referenced by *ObjHandle*. The directory will only be removed if the directory is empty.

### Parameters

<i>ObjHandle</i>	Parent directory object handle.
<i>Path</i>	Specifies the path name of the directory to be removed.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **rmdir**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
EBUSY	The named directory is currently in use and cannot be removed.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	<i>ObjHandle</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

### See Also

**hpss\_Rmdir**, **hpss\_RmdirNoLookup**, **hpss\_Mkdir**, **hpss\_Unlink**.

### Notes

None.



## 2.1.119. hpss\_RmdirNoLookup

### Purpose

Remove an HPSS directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_RmdirNoLookup(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char              *Path,         /* IN */
    sec_cred_t        *Ucred) ;     /* IN */
```

### Description

The **hpss\_RmdirNoLookup** function removes the directory named by *Path* taken relative to the directory referenced by *ObjHandle*. The directory will only be removed if the directory is empty.

### Parameters

<i>ObjHandle</i>	Parent directory object handle.
<i>Path</i>	Specifies the path name of the directory to be removed.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **rmdir**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.
EBUSY	The named directory is currently in use and cannot be removed.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	<i>ObjHandle</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
ENOTEMPTY	The named directory contains entries other than dot and dot-dot.

### See Also

**hpss\_Rmdir**, **hpss\_RmdirHandle**, **hpss\_Mkdir**, **hpss\_Unlink**.

### Notes

This function is different from other **rmdir** calls in that it doesn't make any extra path traverse or get

attribute calls before deleting the specified directory.

## 2.1.120. hpss\_SetACL

### Purpose

Set the Access Control List of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_SetACL(
    char          *Path,          /* IN */
    unsigned32     Options,       /* IN */
    ns_ACLConfArray_t *ACL );    /* IN */
```

### Description

The **hpss\_SetACL** function replaces the access control list for the file named by *Path*.

### Parameters

<i>Path</i>	Names the file for which the <i>ACL</i> is being replaced.
<i>Options</i>	Bit vector used to specify what type of ACL is to be retrieved. One of: HPSS_ACL_OBJECT_ACL – return object's normal ACL HPSS_ACL_INITIAL_OBJECT_ACL – return the initial-object ACL. (only valid for directory or fileset root objects) HPSS_ACL_INITIAL_CONTAINER_ACL – return the initial-container ACL. (only valid for directory or fileset root objects)
<i>ACL</i>	Points to the new access control list for the file.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below:

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer. One of the following has occurred: <ul style="list-style-type: none"><li>● More than one of the HPSS_ACL_OPTION_* bits are set.</li><li>● <i>ObjHandle</i> is NULL.</li><li>● HPSS_ACL_INITIAL_CONTAINER_ACL or HPSS_ACL_INITIAL_OBJECT_ACL was set and the ACL object is not a directory or fileset root.</li></ul>
EINVAL	
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.

ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

#### See Also

**hpss\_SetACLHandle, hpss\_DeleteACL, hpss\_DeleteACLHandle, hpss\_GetACL, hpss\_GetACLHandle, hpss\_UpdateACL, hpss\_UpdateACLHandle**

#### Notes

Exactly one of the HPSS\_ACL\_OPTION\_\* bits must be set in the *Options* bit vector.

## 2.1.121. hpss\_SetACLHandle

#### Purpose

Set the Access Control List of a file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_SetACLHandle(
    ns_ObjHandle_t    *ObjHandle,      /* IN */
    char              *Path,           /* IN */
    sec_cred_t        *Ucred,          /* IN */
    unsigned32         Options,        /* IN */
    ns_ACLConfArray_t *ACL );          /* IN */
```

#### Description

The **hpss\_SetACLHandle** function sets the access control list for the file named by *Path* taken relative to the directory indicated by *ObjHandle*.

#### Parameters

<i>ObjHandle</i>	Parent directory's object handle.
<i>Path</i>	Names the file for which the <i>ACL</i> is being replaced.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>Options</i>	Bit vector used to specify what type of ACL is to be retrieved. One of: HPSS_ACL_OBJECT_ACL - Return object's normal ACL.  HPSS_ACL_INITIAL_OBJECT_ACL - Return the initial-object ACL. (Only valid for directory or fileset root objects)  HPSS_ACL_INITIAL_CONTAINER_ACL - Return the initial-container ACL. (Only valid for directory or fileset root objects)
<i>ACL</i>	Points to the new access control list for the file.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below:

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	One of the following has occurred: <ul style="list-style-type: none"> <li>• More than one of the HPSS_ACL_OPTION_* bits are set.</li> <li>• <i>ObjHandle</i> is NULL.</li> <li>• HPSS_ACL_INITIAL_CONTAINER_ACL or HPSS_ACL_INITIAL_OBJECT_ACL was set and the ACL object is not a directory or fileset root.</li> </ul>
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

#### See Also

**hpss\_SetACL, hpss\_DeleteACL, hpss\_DeleteACLHandle, hpss\_GetACL, hpss\_GetACLHandle, hpss\_UpdateACL, hpss\_UpdateACLHandle**

#### Notes

Exactly one of the HPSS\_ACL\_OPTION\_\* bits must be set in the *Options* bit vector.

## 2.1.122. hpss\_SetAcct

#### Purpose

Change the current account code.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_SetAcct(
    acct_rec_t      NewCurAcct );    /* IN */
```

#### Description

The **hpss\_SetAcct** routine changes the accounting code used when creating files and directories for the current thread.

#### Parameters

<i>NewCurAcct</i>	Specifies the value to be used for the account code for created files and directories.
-------------------	--

#### Return Values

Upon successful completion, **hpss\_SetAcct** returns zero. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below:

#### Error Conditions

EFAULT	The account name with this account code is NULL.
EINVAL	The client is configured for Unix-style accounting, and therefore the account code cannot be changed.
ENOENT	Account doesn't exist.
EPERM	User is not a member of this account.

#### See Also

**hpss\_AcctCodeToName, hpss\_AcctNameToCode, hpss\_Chacct, hpss\_ChacctByName, hpss\_GetAcct, hpss\_GetAcctName, hpss\_SetAcctByName**

#### Notes

None.

## 2.1.123. hpss\_SetAcctByName

#### Purpose

Change the current account name.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_SetAcctByName (
    char          *NewAcctName )    /* IN */
```

#### Description

The **hpss\_SetAcctByName** routine changes the current session account name for the current site for this thread. Since each site contacted by each thread of the Client API will have its own session account, the account name returned is that of the site managing the current working directory.

#### Parameters

<i>NewAcctName</i>	When using site-style accounting, this is the new session account name for the current site. However, when using UID accounting, <i>NewAcctName</i> can have one of several special meanings:  "" – means set the current session account to the UID of the user  String form of an account number (eg. "123") - means change the current session account index to 123.  User name (eg. "smithj") – means use the UID of user "smithj" as the new current session account index.
--------------------	--

#### Return Values

Upon successful completion, **hpss\_SetAcctByName** returns 0. Otherwise, **hpss\_SetAcctByName** returns a

negative value, the absolute value of which indicates the specific error.

## Error Conditions

EFAULT	<i>NewAcctName</i> is a NULL pointer.
ENAMETOOLONG	Length of string pointed to by <i>NewAcctName</i> greater than allowed. (HPSS_MAX_ACCOUNT_NAME – 1)
ENOENT	The account specified by <i>NewAcctName</i> doesn't exist.
EPERM	The user does not have sufficient privilege to change the account name to <i>NewAcctName</i> , or <i>NewAcctName</i> is not defined at the site indicated by the client's current working directory.

## See Also

**hpss\_GetAcct, hpss\_Chacct, hpss\_SetAcct, hpss\_GetAcctName, hpss\_ChacctByName.**

## Notes

If the user's realm ID is foreign, then a message is returned advising that site-style accounting is required. If the *NewAcctName* is null, the the user's uid will be returned. If the *NewAcctName* is a number, then the same number will be returned. Otherwise, the account name is looked up. If the entry is found, then the corresponding uid is returned. If the entry is not found, ENOENT is returned.

## 2.1.124. hpss\_SetAttrHandle

### Purpose

Alter file attribute values.

### Synopsis

```
#include "hpss_api.h"
int
hpss_SetAttrHandle(
    ns_ObjHandle_t      ObjHandle,      /* IN */
    char                *Path,          /* IN */
    sec_cred_t          *Ucred,         /* IN */
    u_signed64          SelFlagsIn,     /* IN */
    hpss_vattr_t        *AttrIn,        /* IN */
    u_signed64          *SelFlagsOut,    /* OUT */
    hpss_vattr_t        *AttrOut );     /* OUT */
```

### Description

The **hpss\_SetAttrHandle** function updates attribute values for the file or directory named by *Path*, taken relative to the directory indicated by *ObjHandle*. The attributes to be updated are indicated by *SelFlagsIn*, and the new attribute values are contained in the structure pointed to by *AttrIn*. Indication of attributes actually changed are returned in *SelFlagsOut*, and the new attribute values in *AttrOut*.

If the input *Path* refers to a symbolic link, **hpss\_SetAttrHandle** returns information about the link itself. The **hpss\_FileSet\*** attributes functions operate on the object to which the link points.

### Parameters

<i>ObjHandle</i>	Parent object handle.
<i>Path</i>	Points to the name of the file for which attribute values are to be changed.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>SelfFlagsIn</i>	Bitmask which indicates which attributes are to be set in the Core Server attributes. Refer to <b>hpss_FileSetAttributes</b> for a list of attributes that can be set using this function.
<i>AttrIn</i>	Points to a structure containing the new attribute values.
<i>SelfFlagsOut</i>	Bitmask indicating the attributes that were set.
<i>AttrOut</i>	<i>Points to a structure that will contain the file attribute values after completion of this request.</i>

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which indicates the specific error.

### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> , <i>AttrIn</i> or <i>AttrOut</i> parameter is a NULL pointer.
EINVAL	An attribute value or selection flag is invalid, or the <i>ObjHandle</i> is NULL.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOSPC	Resources could not be allocated to satisfy the request.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EOPNOTSUPP	The requested change is not supported.
EPERM	The client does not have the appropriate privileges to change the file's attributes.

### See Also

**hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileSetAttributesHandle**, **hpss\_FileSetAttributesSOID**, **hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**.

### Notes

None.

## 2.1.125. hpss\_SetAuditInfo

### Purpose

Set per-thread audit information.

### Synopsis

```
#include "hpss_api.h"
```

```
int
hpss_SetAuditInfo(
    unsigned int      EndUserHostAddr); /* IN */
```

### Description

This routine sets up per-thread information that will appear in Core Server security audit messages.

### Parameters

*EndUserHostAddr*            IP address of end user's host.

### Return Values

Upon successful completion, a value of zero is returned.

### Error Conditions

None.

### See Also

None.

### Notes

This routine should be called by every thread that wants to use per-thread audit information. If the routine is never called, then the audit messages will show the host address of the server (eg., ftpd) that initiated the request. This is typically not very helpful; so the servers should call the routine at least once. If the routine is called once by one thread but not by another thread, then the audit messages will show the host address established by the first thread. This feature lets ftpd call the routine once in the main thread rather than forcing ftpd to call it in every thread that contacts Core Server.

## 2.1.126. hpss\_SetConfiguration

### Purpose

Update the current Client API configuration information.

### Synopsis

```
#include "hpss_api.h"
#include "api_internal.h"
long
hpss_SetConfiguration(
    api_config_t      *ConfigIn );            /* IN */
```

### Description

The **hpss\_SetConfiguration** routine updates the current configuration values for the Client API.

### Parameters

*ConfigIn*                    Points to a structure that contains the new configuration attributes value settings.

### Return Values



Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EFAULT	The <i>ConfigIn</i> parameter is a NULL pointer.
EINVAL	Invalid configuration attribute value setting.

#### See Also

**hpss\_GetConfiguration**, **hpss\_ClientAPIReset**.

#### Notes

None.

## 2.1.127. hpss\_SetCOSByHints

#### Purpose

Set Class Of Service of an open, empty file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_SetCOSByHints (
    int                Filedes,          /* IN */
    unsigned32         Flags,            /* IN */
    hpss_cos_hints_t    *HintsPtr,        /* IN */
    hpss_cos_priorities_t *PrioPtr,      /* IN */
    hpss_cos_md_t       *COSPtr );      /* OUT */
```

#### Description

The **hpss\_SetCOSByHints** routine is used to attempt to place a file in an appropriate Class of Service before any data has been written to that file. This interface is primarily used when the file size is not known at the time the file is created, but based on the knowledge of the file at the time of the first write, a better Class of Service may be determined.

#### Parameters

<i>Filedes</i>	Specifies the open file handle for which the Class of Service is to be set.
<i>Flags</i>	Specifies flags which affect the processing of this request. Valid values are: BFS_RESET_SEGSIZE - Indicates that the request is only to set a new storage segment size.
<i>HintsPtr</i>	Points to a structure that provides attribute values for selection of a new Class of Service or storage segment size.
<i>PrioPtr</i>	Points to a structure that contains relative priorities for the attribute values indicated by the <i>HintsPtr</i> parameter.
<i>COSPtr</i>	Points to a structure that will contain the attribute values for the selected Class of Service and storage segment size.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an `errno` value defined below.

### Error Conditions

EBADF	The specified file descriptor does not refer to an open file.
EBUSY	The file is currently in use by another client thread, or the Core Server could not complete the request at the current time.
EFAULT	One of <i>HinstPtr</i> , <i>PrioPtr</i> , or <i>COSPtr</i> is a NULL pointer.
EINVAL	The specified COS hints are invalid.
EPERM	The client does not have the appropriate privileges to perform the request.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_Open, hpss\_OpenHandle, hpss\_OpenBitfile, hpss\_OpenBitfileVAttrs.**

### Notes

None.

## 2.1.128. hpss\_SetFileOffset

### Purpose

Set the current file offset for an open file, given a 64-bit offset value.

### Synopsis

```
#include <unistd.h>
#include "hpss_api.h"
int
hpss_SetFileOffset(
    int          Fildes,           /* IN */
    u_signed64   OffsetIn,        /* IN */
    int          Whence,           /* IN */
    int          Direction,       /* IN */
    u_signed64   *OffsetOut );    /* OUT */
```

### Description

The **hpss\_SetFileOffset** function sets the file offset for the open file handle, *Fildes*. Refer to the POSIX.1 *lseek* function for more detailed information. Both input and output offset values are 64-bit values, to provide accessibility to the full range of HPSS file sizes. Note that since the *OffsetIn* value is unsigned, the *Direction* parameter is provided to specify whether *OffsetIn* should be used to move forward or backward in the file.

### Parameters

<i>Fildes</i>	Specifies the open file handle for which the file offset is to be set.
<i>OffsetIn</i>	Specifies the number of bytes to be used in calculating the new file offset - dependent on the value of <i>Whence</i> and <i>Direction</i> as to the final effect on the new file offset.

<i>Whence</i>	<p>Specifies how to interpret the <i>OffsetIn</i> parameter in setting the file pointer associated with the <i>Fildes</i> parameter. Values for the <i>Whence</i> parameter are as follows:</p> <p>SEEK_SET - file offset set to <i>OffsetIn</i>.</p> <p>SEEK_CUR - file offset set to current offset plus <i>OffsetIn</i>.</p> <p>SEEK_END - file offset set to current end of file plus <i>OffsetIn</i>.</p> <p>SEEK_CUR - file offset set to current offset plus <i>OffsetIn</i>.</p> <p>SEEK_END - file offset set to current end of file plus <i>OffsetIn</i>.</p>
<i>Direction</i>	<p>Specifies whether <i>OffsetIn</i> should be used to move forward or backward in the file.</p> <p>HPSS_SET_OFFSET_FORWARD - consider <i>OffsetIn</i> as being a nonnegative number.</p> <p>HPSS_SET_OFFSET_BACKWARD - consider <i>OffsetIn</i> as being a negative number.</p>
<i>OffsetOut</i>	Points to an area to contain the file offset as a result of processing this request.

### Return Values

Upon successful completion, **hpss\_SetFileOffset** returns zero. Otherwise **hpss\_SetFileOffset** returns a negative value, the absolute value of which indicates the specific error.

### Error Conditions

EBADF	The specified file descriptor does not refer to an open file.
EBUSY	The file is currently in use by another client thread.
EFAULT	<i>OffsetOut</i> is a NULL pointer.
EINVAL	The <i>Whence</i> or <i>Direction</i> parameter is invalid or the resulting offset would be invalid such as a negative value or beyond the largest supported file size.
ENOSPC	Resources could not be allocated to satisfy the request.

### See Also

**hpss\_Read**, **hpss\_Write**, **hpss\_Lseek**.

### Notes

None.

## 2.1.129. hpss\_SetLoginCred

### Purpose

Establish HPSS login credentials.

### Synopsis

```
#include "hpss_api.h"
signed32
hpss_SetLoginCred(
```

```

char          *PrincipalName,    /* IN */
hpss_authn_mech_t Mechanism,    /* IN */
hpss_rpc_cred_type_t CredType,  /* IN */
hpss_rpc_auth_type_t AuthType,  /* IN */
void          *Authenticator);  /* IN */

```

## Description

This routine is called to set an HPSS login credential. A newly set credential will be refreshed in the background and an **hpss\_PurgeLoginCred** call must be issued to end update of the specified user credentials.

## Parameters

<i>PrincipalName</i>	Specifies the server's principal name.
<i>Mechanism</i>	Security mechanism.
<i>CredType</i>	Type of credentials needed.
<i>AuthType</i>	Authenticator type.
<i>Authenticator</i>	Pointer to the authenticator.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, one of the error conditions below is returned.

## Error Conditions

EINVAL	<i>PrincipalName</i> is a NULL pointer or principle is invalid.
--------	---

## See Also

**hpss\_PurgeLoginCred**

## Notes

None.

## 2.1.130. hpss\_SiteIdToName

### Purpose

Convert a given HPSS site id to its corresponding HPSS site name.

### Synopsis

```

#include "hpss_api.h"
int
hpss_SiteIdToName(
    hpss_uuid_t  *SiteId,    /* IN */
    char         *SiteName ); /* OUT */

```

### Description

The **hpss\_SiteIdToName** routine converts the HPSS site id given in *SiteId* to its corresponding string representation, returned in *SiteName*. When an error is encountered, **hpss\_SiteIdToName** will return

immediately.

#### Parameters

<i>SiteId</i>	If the site id pointer is NULL, or points to a zeroed parameter, the current local id is used.
<i>SiteName</i>	An HPSS site's text name. If the site name is NULL, return the current local site name. Site name should be a string of at least HPSS_MAX_DESC_NAME (32) bytes.

#### Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

ENOENT	The value passed in <i>SiteId</i> does not correspond to a known HPSS site.
ECONN	There was a communication problem during the translation.

#### See Also

**hpss\_SiteNameToId**

#### Notes

None.

## 2.1.131. hpss\_SiteNameToId

#### Purpose

Convert a given HPSS site name to its corresponding HPSS site id.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_SiteNameToId(
    char          *SiteName,      /* IN */
    hpss_uuid_t   *SiteId );     /* OUT */
```

#### Description

The `hpss_SiteNameToId` routine converts the HPSS site name given in *SiteName* to its corresponding HPSS site id, returned in *SiteId*. When an error is encountered, `hpss_SiteNameToId` will return immediately.

#### Parameters

<i>SiteName</i>	An HPSS site's text name. If the site name pointer is null, or points to an empty string, the current local site name is used.
<i>SiteId</i>	A pointer to an HPSS site's UUID. If the site id is null, return the current local site id.

#### Return Values

Upon successful completion, a value of 0 is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

ENOENT	The value passed in <i>SiteName</i> does not correspond to a known HPSS site.
ECONN	There was a communication problem during the translation.

#### See Also

**hpss\_SitIdToName**

#### Notes

None.

## 2.1.132. hpss\_Stage

#### Purpose

Stage a piece of a file to a specified level in the storage hierarchy.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Stage(
    int                Fildes,          /* IN */
    u_signed64         Offset,          /* IN */
    u_signed64         Length,          /* IN */
    unsigned32         StorageLevel,    /* IN */
    unsigned32         Flags );         /* IN */
```

#### Description

The **hpss\_Stage** routine stages part of an open file, specified by *Fildes*, *Offset* and *Length* to a level in the storage hierarchy, specified by *StorageLevel*. The *Flags* argument is used to control behavior of the request.

#### Parameters

<i>Fildes</i>	Specifies the file descriptor, identifying the file to be staged.
<i>Offset</i>	Specifies the offset of the start of the data to be staged.
<i>Length</i>	Specifies the length of the data to be staged.
<i>StorageLevel</i>	Identifies the level in the storage hierarchy to which the data is to be staged. Currently, the only supported value is 0.
<i>Flags</i>	Controls the behavior of the stage request. Valid values include: BFS_STAGE_ALL - stage entire file. BFS_ASYNC_CALL - return after initiating stage.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the

absolute value of which is equal to an errno value defined below.

#### Error Conditions

EBADF	The supplied file descriptor does not correspond to an open file.
EINVAL	The <i>Flags</i> , <i>Offset</i> or <i>Length</i> argument is invalid.
EBUSY	The specified file descriptor is currently in use.
EPERM	The client does not have the appropriate privileges to perform the operation.

#### See Also

**hpss\_Migrate**, **hpss\_Purge**, **hpss\_StageCallback**.

#### Notes

None.

## 2.1.133. hpss\_StageCallback

#### Purpose

Initiate staging a piece of a file in the background.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_StageCallback(
    char                *Path,           /* IN */
    u_signed64          Offset,          /* IN */
    u_signed64          Length,          /* IN */
    unsigned32          StorageLevel,    /* IN */
    bfs_callback_addr_t *CallbackPtr,    /* IN */
    unsigned32          Flags,           /* IN */
    hpss_reqid_t        *ReqID,          /* OUT */
    hpssoid_t           *BitfileID );    /* OUT */
```

#### Description

The **hpss\_StageCallback** routine initiates a background stage of part of a file, specified by *Fildes*, *Offset* and *Length* to a level in the storage hierarchy, specified by *StorageLevel*. The *Flags* argument is used to control behavior of the request.

#### Parameters

<i>Path</i>	Specifies the pathname of the file to be staged.
<i>Offset</i>	Specifies the offset of the start of the data to be staged.
<i>Length</i>	Specifies the amount of the data to be staged.
<i>StorageLevel</i>	Identifies the level in the storage hierarchy to which the data is to be staged.
<i>CallbackPtr</i>	Callback information. If NULL, no callback actions are taken.
<i>Flags</i>	Controls the behavior of the stage request. Valid values include: BFS_STAGE_ALL - stage entire file.

<i>ReqID</i>	Assigned request identification number.
<i>BitfileID</i>	Bitfile ID.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EACCESS	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	The value of the <i>Offset</i> parameter is beyond the end of the file or the <i>StorageLevel</i> parameter is invalid for the storage hierarchy.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the pathname exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_Migrate**, **hpss\_Purge**, **hpss\_Stage**.

### Notes

None.

## 2.1.134. hpss\_Stat

### Purpose

Get file status (POSIX).

### Synopsis

```
#include "hpss_api.h"
int
hpss_Stat(
    char          *Path,          /* IN */
    hpss_stat_t   *Buf );        /* OUT */
```

### Description

The **hpss\_Stat** function obtains information about the file named by *Path* and returns it in the structure pointed to by *Buf*. Refer to POSIX.1 for more detailed information. If *Path* refers to a junction, the junction will be traversed.

### Parameters

<i>Path</i>	Points to the path name of the file being queried.
<i>Buf</i>	Points to a stat structure that will contain the status information for the file.

### Return Values



Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **stat**.

#### Error Conditions

EACCES	Search permission is denied for a component of the path prefix.
EFAULT	The <i>Path</i> or <i>Buf</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

#### See Also

**hpss\_Chown**, **hpss\_Chmod**, **hpss\_Utime**, **hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileGetAttributesSOID**, **hpss\_FileGetXAttributes**, **hpss\_FileSetAttributes**, **hpss\_FileSetAttributesHandle**, **hpss\_FileSetAttributesSOID**, **hpss\_Lstat**, **hpss\_Fstat**, **hpss\_GetListAttrs**, **hpss\_ReadAttrs**.

#### Notes

Note that if the named file is a symbolic link, information is returned for the file to which the contents of the link point. See **hpss\_Lstat** to obtain information about the symbolic link itself.

## 2.1.135. hpss\_Statfs

#### Purpose

Returns file system information for a Class of Service.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Statfs(
    unsigned32      COSId,                /* IN */
    hpss_statfs_t   *StatfsBuffer );     /* OUT */
```

#### Description

The **hpss\_Statfs** routine returns file system information as defined in the statfs structure.

#### Parameters

<i>COSId</i>	Specifies the identifier for the Class of Service that is being queried.
<i>StatfsBuffer</i>	Points to area to contain the file system information.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

#### Error Conditions

EFAULT	The <i>StatfsBuffer</i> parameter is a NULL pointer.
EINVAL	The specified Class of Service does not exist.

#### See Also

None.

#### Notes

1. The block size, *f\_bsize*, is that of the top storage class in the hierarchy.
2. The *f\_name* field of the *statfs* structure will return the site name of the root Core Server.

## 2.1.136. *hpss\_Statvfs*

#### Purpose

Returns file system information for a Class of Service.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Statvfs(
    unsigned32          CosId,          /* IN */
    hpss_statvfs_t      *StatvfsBuffer ); /* OUT */
```

#### Description

The **hpss\_Statvfs** routine returns file system information as defined in the *statvfs* structure for the given Class of Service as a total from all Core Servers which manage files in the Class of Service.

#### Parameters

<i>CosId</i>	The identifier for the Class of Service that is being queried.
<i>StatvfsBuffer</i>	Pointer to area to contain the file system information.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an *errno* value defined below.

#### Error Conditions

EFAULT	The <i>StatvfsBuffer</i> parameter is a NULL pointer.
EINVAL	The specified Class of Service does not exist.

#### See Also

None.

#### Notes

1. The block size, *f\_bsize*, is that of the top storage class in the hierarchy.
2. The *f\_pack* field of the *statvfs* structure will return the site name of the local Core Server that contains the most free space for the given Class of Service.

## 2.1.137. hpss\_Symlink

### Purpose

Create a symbolic link.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Symlink(
    char      *Contents, /* IN */
    char      *Path );  /* IN */
```

### Description

The **hpss\_Symlink** routine creates a symbolic link pointing to the pathname specified in *Contents* with the link's name identified by *Path*.

### Parameters

<i>Contents</i>	Specifies the path name to which the symbolic link will point.
<i>Path</i>	Specifies the name of the symbolic link to be created.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the specified path name.
EFAULT	The <i>Path</i> or <i>Contents</i> parameter is a NULL pointer.
EEXIST	The specified link already exists.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	No entry exists for a component of the path name.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_Readlink, hpss\_ReadlinkHandle, hpss\_SymlinkHandle.**

### Notes

None.

## 2.1.138. hpss\_SymlinkHandle

### Purpose

Create a symbolic link.

## Synopsis

```
#include "hpss_api.h"
int
hpss_SymlinkHandle(
    ns_ObjHandle_t    *ObjHandle,      /* IN */
    char              *Contents,        /* IN */
    char              *Path,            /* IN */
    sec_cred_t        *Ucred,          /* IN */
    hpss_vattr_t      *AttrsOut);      /* OUT */
```

## Description

The **hpss\_SymlinkHandle** function creates a symbolic link with the name *Path* (taken relative to *ObjHandle*), with the contents specified by *Contents*.

## Parameters

<i>ObjHandle</i>	Object handle of existing file.
<i>Contents</i>	Specifies the path name to which the symbolic link will point.
<i>Path</i>	Specifies the name of the symbolic link to be created.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>AttrsOut</i>	Returned attributes of new symbolic link.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the specified path name.
EFAULT	The <i>Path</i> or <i>Contents</i> parameter is a NULL pointer.
EEXIST	The specified link already exists.
EINVAL	<i>ObjHandle</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	No entry exists for a component of the path name.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Readlink, hpss\_ReadlinkHandle, hpss\_Symlink.**

## Notes

None.

## 2.1.139. hpss\_ThreadCleanUp

### Purpose

Cleans up a thread's Client API state.

### Synopsis

```
#include "hpss_api.h"
int
hpss_ThreadCleanUp(
    pthread_t      ThreadID );      /* IN */
```

### Description

The **hpss\_ThreadCleanUp** routine frees resources used by a thread's Client API context.

### Parameters

*ThreadID*                      Specifies the thread identifier for the thread whose resources are to be freed.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

ENOENT                      State for the specified thread could not be found.

### See Also

None.

### Notes

The **hpss\_ThreadCleanUp** routine should be called once for each thread which terminates and has previously called the Client API.

## 2.1.140. hpss\_Truncate

### Purpose

Set the length of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_Truncate(
    char          *Path,          /* IN */
    u_signed64    Length );      /* IN */
```

### Description

The **hpss\_Truncate** routine sets the length of a file, specified by the *Path* argument. If the new file length is less than the current length, the space allocated beyond the new length will be freed. If the new length is greater than the current length, a hole is created in the file.

## Parameters

<i>Path</i>	Specifies the path name of the file to be truncated.
<i>Length</i>	Specifies the new length of the file.

## Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

## Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	<i>Path</i> specifies a directory.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The specified path name does not exist.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_TruncateHandle**, **hpss\_Ftruncate**, **hpss\_Fclear**, **hpss\_FileSetAttributes**.

## Notes

None.

## 2.1.141. hpss\_TruncateHandle

### Purpose

Set the length of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_TruncateHandle(
    ns_ObjHandle_t    ObjHandle,      /* IN */
    char               *Path,          /* IN */
    u_signed64         Length,         /* IN */
    sec_cred_t         Ucred);        /* IN */
```

### Description

The **hpss\_TruncateHandle** function sets the file length for the file specified by *ObjHandle*.

### Parameters

<i>ObjHandle</i>	Object file handle.
<i>Path</i>	Specifies the path name of the file to be truncated.

<i>Length</i>	Specifies the new length of the file.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the file.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	<i>Path</i> specifies a directory or <i>ObjHandle</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The specified path name does not exist.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

### See Also

**hpss\_Truncate, hpss\_Ftruncate, hpss\_Fclear, hpss\_FileSetAttributes.**

### Notes

None.

## 2.1.142. hpss\_Umask

### Purpose

Set the file creation mask.

### Synopsis

```
#include "hpss_api.h"
mode_t
hpss_Umask (
    mode_t      Cmask );          /* IN */
```

### Description

The **hpss\_Umask** function sets the file mode creation mask of the thread and returns the previous value of the mask. Refer to POSIX.1 **umask** for further details.

### Parameters

<i>Cmask</i>	Specifies the file mode creation mask to be used by subsequent <b>hpss_Open, hpss_OpenHandle, hpss_Create, hpss_CreateHandle, hpss_CreateDMHandle, hpss_Mkdir, and hpss_MkdirHandle</b> calls.
--------------	--

### Return Values

**hpss\_Umask** returns the previous file mode creation mask for the thread.

#### Error Conditions

The **hpss\_Umask** function is always successful and no return values are reserved to indicate an error.

#### See Also

**hpss\_Open**, **hpss\_OpenHandle**, **hpss\_Create**, **hpss\_CreateHandle**, **hpss\_Mkdir**, **hpss\_MkdirHandle**.

#### Notes

None.

## 2.1.143. hpss\_Unlink

#### Purpose

Remove an entry from an HPSS directory.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_Unlink(
    char          *Path );          /* IN */
```

#### Description

The **hpss\_Unlink** function removes the entry named by the *Path*, and decrements the link count of the file. If the link count becomes zero, the file will be deleted when it is no longer open by any client.

#### Parameters

*Path*                      Names the directory entry to be removed.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **unlink**.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The file named by <i>Path</i> is a directory.

#### See Also

**hpss\_Close**, **hpss\_UnlinkHandle**, **hpss\_Link**, **hpss\_LinkHandle**.



## Notes

1. Note that using **hpss\_Unlink** to remove directory names is not supported.
2. Also note that if the *Path* refers to a symbolic link, the link itself shall be removed.

## 2.1.144. hpss\_UnlinkHandle

### Purpose

Remove an entry from an HPSS directory.

### Synopsis

```
#include "hpss_api.h"
int
hpss_UnlinkHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char              *Path,          /* IN */
    sec_cred_t        *Ucred) ;      /* IN */
```

### Description

The **hpss\_UnlinkHandle** function removes the entry name by *Path*, taken relative to the directory indicated by *ObjHandle*, and decrements the link count for the file. If link count becomes zero, the file will be deleted when it is no longer open to any client.

### Parameters

<i>ObjHandle</i>	Parent object handle.
<i>Path</i>	Names the directory entry to be removed.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **unlink**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the directory containing the link to be removed.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	<i>ObjHandle</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The file named by <i>Path</i> is a directory.

### See Also

**hpss\_Close, hpss\_Link, hpss\_LinkHandle, hpss\_Unlink.**

#### Notes

1. Note that using **hpss\_Unlink** to remove directory names is not supported.
2. Also note that if the *Path* refers to a symbolic link, the link itself shall be removed.

## 2.1.145. hpss\_UnlinkNoLookup

#### Purpose

Remove an HPSS file.

#### Synopsis

```
#include "hpss_api.h"
int
hpss_UnlinkNoLookup(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char               *Path,        /* IN */
    sec_cred_t        *Ucred) ;     /* IN */
```

#### Description

The **hpss\_UnlinkNoLookup** function removes the file named by *Path* taken relative to the directory referenced by *ObjHandle*.

#### Parameters

<i>ObjHandle</i>	Parent directory object handle.
<i>Path</i>	Specifies the path name of the file to be removed.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.

#### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **unlink**.

#### Error Conditions

EACCES	Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the file to be removed.
EBUSY	The named file is currently in use and cannot be removed.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
EINVAL	<i>ObjHandle</i> is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.

## See Also

**hpss\_Unlink, hpss\_UnlinkHandle, hpss\_Rmdir.**

## Notes

This function is different from other unlink calls in that it doesn't do any extra path traversals or get attribute calls before deleting the specified file.

## 2.1.146. hpss\_UpdateACL

### Purpose

Update entries in the Access Control List of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_UpdateACL(
    char          *Path,          /* IN */
    unsigned32    Options,        /* IN */
    ns_ACLConfArray_t *ACL );    /* IN */
```

### Description

The **hpss\_UpdateACL** function updates entries, specified by *ACL*, in the access control list for the file named by *Path*.

### Parameters

<i>Path</i>	Names the file for which the <i>ACL</i> is being updated.
<i>Options</i>	<p>A bit vector containing bits which control the behavior of <b>hpss_UpdateACL</b>. It is possible to specify which ACL is to be updated, and to specify the behavior of <b>hpss_UpdateACL</b> while calculating the MASK_OBJ.</p> <p><i>Options</i> can be used to mimic the behavior of the following <b>acl_edit</b> options: -n, -c, and -p. This mimicking is done using the following mutually-exclusive constants:</p> <p>HPSS_ACL_DONT_CALC_MASK</p> <p>Specifies that a new MASK_OBJ should not be calculated. This option is useful when the ACL operations require the calculation of a new MASK_OBJ, but doing so would result in an error. This option allows the operations to be carried out, but a new MASK_OBJ is not calculated.</p> <p>HPSS_ACL_CALC_MASK_IGNORE_ERRORS</p> <p>Creates or modifies the object's MASK_OBJ entry with permissions equal to the union of all entries other than type USER_OBJ, OTHER_OBJ, and UNAUTHENTICATED. This creation or modification is done after all other modifications to the ACL are performed. The new MASK_OBJ is set even if it grants permissions previously masked out. It is recommended that this option be used only if not specifying it results in an error. This option is useful only for objects that support the MASK_OBJ entry type and are required to recalculate</p>

a new MASK\_OBJ after they are modified.

#### HPSS\_ACL\_PURGE\_MASKED\_PERMS

Purges all masked permissions (before any other modifications are made). This option is useful only for ACLs that contain an entry of type MASK\_OBJ. Use it to prevent unintentionally granting permissions to an existing entry when a new MASK\_OBJ is calculated as a result of adding or modifying an ACL entry.

*Options* can also be used to specify which ACL is to be updated. The following mutually-exclusive constants can be used to make this selection:

HPSS\_ACL\_OBJECT\_ACL

HPSS\_ACL\_INITIAL\_CONTAINER\_ACL

HPSS\_ACL\_INITIAL\_OBJECT\_ACL

If an update operation creates a MASK\_OBJ that unintentionally adds permissions to an existing ACL entry, the modification causing the MASK\_OBJ recalculation will abort with an error unless the HPSS\_ACL\_CALC\_MASK\_IGNORE\_ERRORS or HPSS\_ACL\_DONT\_CALC\_MASK options are specified.

*ACL*

Points to the ACL entries to be updated.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	There are two sets of mutually exclusive flags available through the <i>Options</i> parameter. Some invalid combination of flags was provided.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

### See Also

**hpss\_UpdateACLHandle, hpss\_DeleteACL, hpss\_DeleteACLHandle, hpss\_GetACL, hpss\_GetACLHandle, hpss\_SetACL, hpss\_SetACLHandle.**

### Notes

None.

## 2.1.147. hpss\_UpdateACLHandle

### Purpose

Update entries in the Access Control List of a file.

### Synopsis

```
#include "hpss_api.h"
int
hpss_UpdateACLHandle(
    ns_ObjHandle_t    *ObjHandle,    /* IN */
    char              *Path,         /* IN */
    sec_cred_t        *Ucred,        /* IN */
    unsigned32         Options,       /* IN */
    ns_ACLConfArray_t *ACL );        /* IN */
```

### Description

The **hpss\_UpdateACLHandle** function updates entries, specified by *ACL*, in the access control list for the file named by *Path* taken relative to the directory indicated by *ObjHandle*.

### Parameters

<i>ObjHandle</i>	Parent object directory handle.
<i>Path</i>	Names the file for which the <i>ACL</i> is being updated.
<i>Ucred</i>	User credentials. If NULL, the credentials in the current thread context are used.
<i>Options</i>	<p>A bit vector containing bits which control the behavior of <b>hpss_UpdateACL</b>. It is possible to specify which ACL is to be updated, and to specify the behavior of <b>hpss_UpdateACLHandle</b> while calculating the MASK_OBJ.</p> <p><i>Options</i> can be used to mimic the behavior of the following <b>acl_edit</b> options: -n, -c, and -p. This mimicking is done using the following mutually-exclusive constants:</p> <p>HPSS_ACL_DONT_CALC_MASK</p> <p>Specifies that a new MASK_OBJ should not be calculated. This option is useful when the ACL operations require the calculation of a new MASK_OBJ, but doing so would result in an error. This option allows the operations to be carried out, but a new MASK_OBJ is not calculated.</p> <p>HPSS_ACL_CALC_MASK_IGNORE_ERRORS</p> <p>Creates or modifies the object's MASK_OBJ entry with permissions equal to the union of all entries other than type USER_OBJ, OTHER_OBJ, and UNAUTHENTICATED. This creation or modification is done after all other modifications to the ACL are performed. The new MASK_OBJ is set even if it grants permissions previously masked out. It is recommended that this option be used only if not specifying it results in an error. This option is useful only for objects that support the MASK_OBJ entry type and are required to recalculate a new MASK_OBJ after they are modified.</p>

## HPSS\_ACL\_PURGE\_MASKED\_PERMS

Purges all masked permissions (before any other modifications are made). This option is useful only for ACLs that contain an entry of type MASK\_OBJ. Use it to prevent unintentionally granting permissions to an existing entry when a new MASK\_OBJ is calculated as a result of adding or modifying an ACL entry.

*Options* can also be used to specify which ACL is to be updated. The following mutually-exclusive constants can be used to make this selection:

HPSS\_ACL\_OBJECT\_ACL

HPSS\_ACL\_INITIAL\_CONTAINER\_ACL

HPSS\_ACL\_INITIAL\_OBJECT\_ACL

*If an update operation creates a MASK\_OBJ that unintentionally adds permissions to an existing ACL entry, the modification causing the MASK\_OBJ recalculation will abort with an error unless the HPSS\_ACL\_CALC\_MASK\_IGNORE\_ERRORS or HPSS\_ACL\_DONT\_CALC\_MASK options are specified.*

*ACL*

Points to the ACL entries to be updated.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value defined below.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> or <i>ACL</i> parameter is a NULL pointer.
EINVAL	There are two sets of mutually exclusive flags available through the <i>Options</i> parameter. Some invalid combination of flags was provided.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

### See Also

**hpss\_UpdateACL, hpss\_DeleteACL, hpss\_DeleteACLHandle, hpss\_GetACL, hpss\_GetACLHandle, hpss\_SetACL, hpss\_SetACLHandle.**

### Notes

None.

## 2.1.148. hpss\_Utime

### Purpose

Set access and modification times of an HPSS file.

### Synopsis

```
#include <utime.h>
#include "hpss_api.h"
int
hpss_Utime(
    char          *Path,          /* IN */
    const struct utimbuf *Times); /* IN */
```

### Description

The **hpss\_Utime** function sets the access and modification times of the file named by *Path* to the values specified in the structure pointed to by *Times*. Refer to POSIX.1 for more detailed information.

### Parameters

<i>Path</i>	Names the file for which times are being changed.
<i>Times</i>	Points to a structure containing the new time values.

### Return Values

Upon successful completion, a value of zero is returned. Otherwise, a negative value is returned, the absolute value of which is equal to an errno value set by POSIX.1 **utime**.

### Error Conditions

EACCES	Search permission is denied on a component of the path prefix.
EFAULT	The <i>Path</i> parameter is a NULL pointer.
ENAMETOOLONG	The length of the <i>Path</i> argument exceeds the system-imposed limit, or a component of the path name exceeds the system-imposed limit.
ENOENT	The named file does not exist, or the <i>Path</i> argument points to an empty string.
ENOTDIR	A component of the <i>Path</i> prefix is not a directory.
EPERM	The client does not have the appropriate privileges to perform the operation.

### See Also

**hpss\_Stat**, **hpss\_Statfs**, **hpss\_Statvfs**, **hpss\_FileGetAttributes**, **hpss\_FileGetAttributesHandle**, **hpss\_FileSetAttributes**, **hpss\_FileSetAttributesHandle**, **hpss\_SetAttrHandle**.

### Notes

None.

## 2.1.149. hpss\_Write

### Purpose

Write data from a client buffer to a contiguous section of an HPSS file, beginning at the current file offset.

### Synopsis

```
#include "hpss_api.h"
```

```

ssize_t
hpss_Write(
    int          Fildes,          /* IN */
    const void    *Buf,          /* IN */
    size_t        Nbyte );      /* IN */

```

## Description

The **hpss\_Write** function attempts to write *Nbyte* bytes from the client buffer pointed to by *Buf* to the file associated with the open file handle, *Fildes*.

## Parameters

<i>Fildes</i>	Specifies the open file handle associated with the file to which data is to be written.
<i>Buf</i>	Points to a buffer where the data is to be found.
<i>Nbyte</i>	Specifies the number of bytes to be written.

## Return Values

Upon successful completion, **hpss\_Write** returns a nonnegative value that is the number of bytes written. Otherwise, **hpss\_Write** returns a negative value; the absolute value of which is equal to an errno value set by POSIX.1 **write**.

## Error Conditions

EBADF	The specified file descriptor does not correspond to a file opened for writing.
EBUSY	The file is currently in use by another client thread.
EFAULT	The <i>Buf</i> parameter is out of range.
EFBIG	The write operation would cause the file to exceed the system-imposed maximum file length.
EIO	An input/output or HPSS internal error occurred.
ENOSPC	There is no free space remaining to satisfy the write request.

## See Also

**hpss\_Open**, **hpss\_OpenHandle**, **hpss\_OpenBitfile**, **hpss\_Read**, **hpss\_Lseek**, **hpss\_SetFileOffset**, **hpss\_ReadList**, **hpss\_WriteList**.

## Notes

None.

## 2.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided. Note: Descriptions of the IOD and IOR structures may be found in Chapter 3.



## 2.2.1. Account Record - acct\_rec\_t

### Description

The account record contains the HPSS account identifier number.

### Format

The API configuration structure has the following format:

```
typedef unsigned32 acct_rec_t;
```

## 2.2.2. API Configuration Structure - api\_config\_t

### Description

The API configuration structure contains values that control optional features of the Client API configuration.

### Format

The API configuration structure has the following format:

```
typedef struct api_config {
    long          Flags;
    long          DebugValue;
    long          TransferType;
    long          NumRetries;
    int           BusyDelay;
    int           BusyRetries;
    int           TotalDelay;
    int           LimitedRetries;
    long          MaxConnections;
    long          ReuseDataConnections;
    long          UsePortRange;
    long          RetryStageInp;
    int           DMAPWriteUpdates;
    hpss_authn_mech_t AuthnMech;
    hpss_rpc_prot_level_t RPCProtLevel;
    char          DescName[HPSS_MAX_DESC_NAME];
    char          DebugPath[HPSS_MAX_PATH_NAME];
    char          HostName[HPSS_MAX_HOST_NAME];
} api_config_t;
```

### Flags

Contains a bitmap of configuration flags. Valid values include:

API_ENABLE_LOGGING	If logging compiled into Client API library, perform HPSS logging on errors.
API_USE_CONFIG	Use configuration from API.
API_DISABLE_CROSS_REALM	If set, this flag prevents the Client API from contacting any servers outside of the local realm. Once set, this flag cannot be unset. This flag is set automatically when FTP logs in without security.

API\_DISABLE\_JUNCTIONS      If set, this flag prevents the Client API from processing any requests which require it to traverse a junction. Once set, this flag cannot be unset. NFS will always set this flag explicitly.

### **DebugValue**

If zero, indicates that Client API will not send debug messages to an output file; otherwise messages will be sent (note that all debug messages are conditionally compiled into the library).

### **TransferType**

Indicates what data transfer mechanism is to be used for transfers handled by the Client API.

API\_TRANSFER\_TCP      Use TCP/IP.

API\_TRANSFER\_IPI3      Use IPI3 protocol if support compiled in.

API\_TRANSFER\_MVRSELECT      Let mover select transfer protocol.

### **NumRetries**

Used to control the number of retries to attempt when an operation fails. Currently this class of operation includes library initialization and communication failures. A value of zero indicates that no retries are to be performed and a value of negative one indicates that operation will be retried until successful.

### **BusyDelay**

Used to control the number of seconds to delay between retry attempts.

### **BusyRetries**

Used to control the number of retries to be performed when a request fails because the Core Server does not currently have an available thread to handle that request. A value of zero indicates that no retries are to be performed. A value of negative one indicates that retries should be attempted until either the request succeeds or fails for another reason.

### **TotalDelay**

Used to control the number of total seconds to continue retrying a request.

### **LimitedRetries**

Used to control the number of retry attempts for limited retry type errors.

### **MaxConnections**

Maximum number of connections for use by the HPSS connection management service.

### **ReuseConnections**

Used to control whether TCP/IP connections are to be left as long as a file is opened or closed after each read or write request. A non-zero value will cause connections to remain open, while a zero will cause connections to be closed.

### **UsePortRange**

Used to control whether the HPSS Mover(s) should use the configured port range when making TCP/IP connections for read and write requests. A non-zero value will cause the Mover(s) to use the port range. A value of zero will cause the Mover(s) to allow the operating system to select the port number.

### **RetryStageInp**

Used to control whether retries are attempted on opens of files in a Class of Service that is configured for background staging on open. A non-zero value indicates that open which would return -EINPROGRESS to indicate the file is being staged will be retried. A value of zero indicates that the -EINPROGRESS return code will be returned to the client.

#### **DMAPWriteUpdates**

Controls the frequency of cache invalidates that are issued to the XD SM file system.

#### **AuthnMech**

Authentication Mechanism to use (Unix or Keberos).

#### **RPCProtLevel**

Level of RPC protection to use.

#### **DescName**

Name to use when generating HPSS log messages.

#### **DebugPath**

If generation of debug messages is enabled, the pathname of the file to which log messages will be directed. Special cases are "stdout" and "stderr".

#### **HostName**

Specifies the interface name to use for TCP/IP communications.

## **2.2.3. API Distributed File Information - api\_dist\_file\_info\_t**

### **Description**

The API Distributed File Information structure contains required information defining a distributed file.

### **Format**

The api\_dist\_file\_info\_t structure has the following format:

```
typedef struct {
    struct file_info {
        hpss_object_handle_t bitfile_handle;
        hpss_uuid_t          CoreServerUUID;
        u_signed64           Offset;
        int                  OpenFlag;
        unsigned32           FilesetCOS;
        api_dmap_attrs_t     DMattrs;
    } Info;
    uchar CkSum;
} api_dist_file_info_t;
```

#### **bitfile\_handle**

The HPSS bitfile object handle.

#### **CoreServerUUID**

The HPSS server universal identifier.

### **Offset**

Offset of the file from the beginning (bytes).

### **OpenFlag**

Specifies the file status and file access modes to be assigned. Applicable values given below may be OR'ed together. Refer to POSIX.1 for specific behavior.

O\_RDONLY

O\_WRONLY

O\_RDWR

O\_APPEND

O\_CREAT

O\_EXCL

O\_TRUNC

### **FilesetCOS**

The Class of Service assigned to the file.

### **DMAttrs**

The DMAP attributes of the file. See the `api_dmap_attrs_t` structure.

### **CkSum**

The computed checksum for the file.

## **2.2.4. API DMAP Attributes - `api_dmap_attrs_t`**

### **Description**

The API DMAP Attributes structure maintains related information concerning a distributed file.

### **Format**

This structure has the following format:

```
typedef struct {
    u_signed64      FilesetID;
    unsigned32      FilesetType;
    hpss_uuid_t     DMGuid;
    unsigned32      HandleLength;
    byte            Handle[HPSS_MAX_DMEPI_HANDLE_SIZE];
} api_dmap_attrs_t;
```

### **FilesetID**

The identifier for the DMAP fileset for the file.

### **FilesetType**

The following fileset types are defined:

NS\_FS\_TYPE\_BACKEDUP

NS\_FS\_TYPE\_NATIVE

NS\_FS\_TYPE\_MIRRORED

NS\_FS\_TYPE\_NS\_LOCAL

#### **DMGuid**

The DM Gateway universal identifier.

#### **HandleLength**

Length in bytes of the DMAP file handle. Maximum currently set at 32 bytes.

#### **Handle**

DMAP file handle.

## 2.2.5. API Name Specification - api\_namespec\_t

### **Description**

The API Name Specification structure is used for converting HPSS realm and principal ids to/from their associated names. The API Name specification structure contains the translation type along with the principal and realm information that is to be converted.

### **Format**

The api\_namespec\_t has the following format:

```
typedef struct api_namespec {  
    namespec_type_t    Type;  
    unsigned32         Id;  
    unsigned32         RealmId;  
    char               Name[HPSS_MAX_PRINCIPLE_NAME] ;  
    char               RealmName[HPSS_MAX_REALM_NAME] ;  
} api_namespec_t;
```

#### **Type**

The type of translation that is requested. The valid values are:

NAMESPEC_REALM	Translate realm name only.
NAMESPEC_USER	Translate user name and realm name.
NAMESPEC_GROUP	Translate group name and realm name.
NAMESPEC_SKIP	Do not translate this entry

#### **Id**

The uid or gid of the principal.

#### **RealmId**

The HPSS realm ID where the principal resides.

**Name**

The name of the principal.

**RealmName**

The name of the realm where the principal resides.

## 2.2.6. Bitfile Volatile and Metadata Attributes - bf\_attrib\_t

### Description

The attributes structure for the bitfile object contains all the volatile and metadata bitfile attributes. These are parameters relating to a bitfile.

### Format

The bitfile attributes structure has the following format:

```
struct bf_attrib {
    u_signed64      CurrentPosition;
    signed32        OpenCount;
    unsigned32      FamilyId;
    bf_attrib_md_t  BfAttribMd;
};
typedef struct bf_attrib bf_attrib;
typedef bf_attrib bf_attrib_t;
```

**CurrentPosition**

Specifies the current byte position in the bitfile.

**OpenCount**

Specifies the current number of clients that have the bitfile open.

**FamilyId**

Specifies the family identifier for the bitfile.

**BfAttribMd**

Specifies the structure of bitfile metadata attributes that are stored in the data base.

## 2.2.7. Bitfile Volatile and Metadata Extended Attributes - bf\_xattrib\_t

### Description

The attributes structure for the bitfile object contains all the volatile and metadata bitfile extended attributes. These are parameters relating to a bitfile and location of valid data.

### Format

The bitfile extended attributes structure has the following format:

```
struct bf_xattrib {
    u_signed64      CurrentPosition;
```

```

        signed32      OpenCount;
        unsigned32    FamilyId;
        bf_sc_attr_t  SCAttrib[HPSS_MAX_STORAGE_LEVELS];
        bf_attr_md_t  BfAttribMd;
};
typedef struct bf_xattrib bf_xattrib;
typedef bf_xattrib bf_xattrib_t;

```

#### **CurrentPosition**

Specifies the current byte position in the bitfile.

#### **OpenCount**

Specifies the current number of clients that have the bitfile open.

#### **FamilyId**

Specifies the family identifier for the bitfile.

#### **SCAttrib**

Specifies the storage class attributes at each valid level in the hierarchy.

#### **BfAttribMd**

Specifies the structure of bitfile metadata attributes that are stored in the data base.

## **2.2.8. Bitfile Metadata Attributes - bf\_attr\_md\_t**

### **Description**

This structure contains the bitfile attributes metadata. These are parameters relating to a bitfile.

*LinkCount* is always 1 for an existing bitfile in current HPSS release. On one bfs\_Bitfile(Open)SetAttrs call, reverse maps (OwnerRec) can be either added or deleted. Both cannot be accomplished on the same call.

### **Format**

The bitfile attributes metadata structure has the following format:

```

struct bf_attr_md {
    u_signed64      DataLen;
    signed32        ReadCount;
    signed32        WriteCount;
    signed32        LinkCount;
    timestamp_sec_t CreateTime;
    timestamp_sec_t ModifyTime;
    timestamp_sec_t WriteTime;
    timestamp_sec_t ReadTime;
    unsigned32      COSId;
    unsigned32      NewCOSId;
    acct_rec_t      Acct;
    unsigned32      Flags;
    unsigned32      StorageSegMult;
    u_signed64      RegisterBitmap;
    unsigned32      RealmId;
};

```

```
typedef struct bf_attrb_md bf_attrb_md;
typedef bf_attrb_md bf_attrb_md_t;
```

### **DataLen**

Specifies the number of bytes of actual data that the bitfile contains.

### **ReadCount**

Specifies the count of the number of times that all or part of the bitfile has been read.

### **WriteCount**

Specifies the count of the number of times that data has been written to the bitfile.

### **LinkCount**

Specifies the number of links to this bitfile.

### **CreateTime**

Specifies the date and time the bitfile was created.

### **ModifyTime**

Specifies the date and time the bitfile was last modified.

### **WriteTime**

Specifies the date and time when data was last written to the bitfile.

### **ReadTime**

Specifies the date and time when the bitfile was last read.

### **COSId**

Specifies the class of service type (unsigned32) and indicates which of several classes of service the bitfile is in. This ID references a class of service record that defines the parameters for this particular class of service. When changing a file's COS, this field is used by the `hpss_FileSetAttributes` function.

### **NewCOSId**

Indicates the new class of service that a file is to be changed to when the client changes the class of service on a bitfile. When the change has been completed, the value of this field is moved into `COSId` and this field is cleared. This field is read only. Use the `COSId` field to change a file's COS.

### **Acct**

Specifies the accounting metadata for the bitfile. It includes information needed to charge for data storage, access, transfers, quotas, etc.

### **Flags**

Used to maintain boolean data related to the bitfile. The following constants correspond to flag bits:

<code>SYNC_FILESET_DATA</code>	Make consistency check
<code>HPSS_DATA_VALID</code>	HPSS data is valid
<code>CACHE_DATA_VALID</code>	File system cache is valid



BFS_ACCT_STATS_VALID1	Accounting stats are valid, bit 1
BFS_ACCT_STATS_VALID2	Accounting stats are valid, bit 2
BFS_NO_PURGE	Bitfile cannot be purged

### **StorageSegMult**

Storage segment multiple used to adjust size of disk storage segments.

### **RegisterBitmap**

This bit vector is used to indicate which fields in the attribute structure the SSM wants to receive notifications about when the field changes. The following constants can be used to set the bits to indicate which fields are to be monitored:

BFS\_REG\_OPEN\_COUNT  
 BFS\_REG\_DATA\_LEN  
 BFS\_REG\_READ\_COUNT  
 BFS\_REG\_WRITE\_COUNT  
 BFS\_REG\_LINK\_COUNT  
 BFS\_REG\_CREATE\_TIME  
 BFS\_REG\_MODIFY\_TIME  
 BFS\_REG\_WRITE\_TIME  
 BFS\_REG\_READ\_TIME  
 BFS\_REG\_COS\_ID  
 BFS\_REG\_ACCT  
 BFS\_REG\_REALM\_ID  
 BFS\_REG\_FAMILY\_ID

### **RealmId**

The client realm identifier.

## **2.2.9. Bitfile Service Storage Class Attributes - bf\_sc\_attr\_t**

### **Description**

This structure contains storage class information for a specific storage hierarchy level at which the specified bitfile exists.

### **Format**

The storage class attributes have the following format:

```

struct bf_sc_attr {
    bf_vv_attr_t  VVAttr[BFS_MAX_VV_TO_RETURN_AT_LEVEL];
    unsigned32    NumberOfVVs;
    u_signed64    BytesAtLevel;
    unsigned32    OptimumAccessSize;
    unsigned32    StripeWidth;
}
  
```

```

        u_signed64      StripeLength;
        unsigned32      Flags;
};
typedef struct bf_sc_attr bf_sc_attr;
typedef bf_sc_attr bf_sc_attr_t;

```

#### **VVAttrib**

An array of virtual volumes on which bitfile segments are contained.

#### **NumberOfVVs**

Specifies the number of virtual volume entries in the array.

#### **BytesAtLevel**

Specifies the amount of data that exist at this level (in bytes).

#### **OptimumAccessSize**

Specifies the optimum access size of the storage class.

#### **StripeWidth**

Specifies the stripe width of the storage class.

#### **StripeLength**

Specifies the stripe length of the storage class.

#### **Flags**

The flags that defined the state of the fileset. Valid values include:

BFS_BFATTRS_LEVEL_IS_DISK	This is a disk storage level.
BFS_BFATTRS_LEVEL_IS_TAPE	This is a tape storage level.
BFS_BFATTRS_DATAEXISTS_AT_LEVEL	This is a tape storage level.
BFS_BFATTRS_ADDITIONAL_VV_EXIST	More VVs exist than the maximum the structure can hold.

## **2.2.10. Bitfile Service Virtual Volume Attributes - bf\_vv\_attr\_t**

### **Description**

This structure contains Core Server virtual volume attributes for a specific storage level in the hierarchy.

### **Format**

The bitfile virtual volume attributes have the following format:

```

struct bf_vv_attr {
    hpssoid_t      VVID;
    signed32       RelPosition;
    u_signed64     BytesOnVV;
    pv_list_t      *PVList;
}

```

```
};
typedef struct bf_vv_attr bf_vv_attr;
typedef bf_vv_attr bf_vv_attr_t;
```

### **VVID**

Specifies the virtual volume identifier.

### **RelPosition**

Specifies the relative start position of first bitfile segment on this virtual volume.

### **BytesOnVV**

Specifies the number of bytes of the given file stored on this VV from the current contiguous set of bitfile segments. It is possible for BytesOnVV to be less than the total number of bytes of the given file that reside on this VV if non- adjacent bitfile segments are stored on the same VV. For example, if a file consists of three 1MB bitfile segments, with the first and third on VV 'A' and the second on VV 'B', an hpss\_FileGetXAttributes() call will return a list of three VVs (bf\_vv\_attr\_t types), each with BytesOnVV = 1MB. On the other hand, if a file consists of three 1MB bitfile segments, with the first and second on VV 'A' and the third on VV 'B' an hpss\_FileGetXAttributes() call will return a list of two VVs, the first with BytesOnVV = 2MB and the second with BytesOnVV = 1MB.

### **PVList**

A conformant array of physical volume attributes.

## **2.2.11. Bitfile Callback Address - bfs\_callback\_addr\_t**

### **Description**

The Bitfile Callback Address structure contains the host information, port number and an identification number which facilitates call backs during a stage process.

### **Format**

The bfs\_callback\_addr\_t has the following format:

```
struct bfs_callback_addr {
    unsigned32    addr;
    unsigned16    port;
    unsigned16    family;
    signed32      id;
};
typedef struct bfs_callback_addr bfs_callback_addr;
typedef bfs_callback_addr bfs_callback_addr_t;
```

### **addr**

Host address in network byte order.

### **port**

Port number.

### **family**

Address family.

**Id**

Id to be returned during a callback.

## 2.2.12. Core Server Attribute Structure - hpss\_Attrs\_t

### Description

The Core Server external attribute structure contains fields for the various attributes (metadata) that the Core Server maintains for an object.

### Format

The Core Server external attributes structure has the following format:

```
struct hpss_Attrs {
    acct_rec_t      Account;
    hpsoid_t        BitfileId;
    unsigned32      RealmId;
    char            Comment[256];
    unsigned32      CompositePerms;
    unsigned32      COSId;
    u_signed64      DataLength;
    unsigned32      DMDDataStateFlags;
    byte            DMHandle[HPSS_MAX_DMEPI_HANDLE_SIZE];
    unsigned32      DMHandleLength;
    unsigned32      DontPurge;
    unsigned32      EntryCount;
    unsigned32      ExtendedACLs;
    unsigned32      FamilyId;
    ns_ObjHandle_t  FilesetHandle;
    u_signed64      FilesetId;
    u_signed64      FilesetRootId;
    unsigned32      FilesetStateFlags;
    unsigned32      FilesetType;
    hpss_uuid_t     GatewayUUID;
    unsigned32      GID;
    unsigned32      GroupPerms;
    unsigned32      LinkCount;
    unsigned32      MACSecLabel;
    unsigned32      OpenCount;
    unsigned32      OtherPerms;
    unsigned32      ReadCount;
    u_signed64      RegisterBitMap;
    unsigned32      SetGIDBit;
    unsigned32      SetStickyBit;
    unsigned32      SetUIDBit;
    unsigned32      SubSystemId;
    timestamp_sec_t TimeCreated;
    timestamp_sec_t TimeLastRead;
    timestamp_sec_t TimeLastWritten;
    timestamp_sec_t TimeModified;
    unsigned32      Type;
```

```

        unsigned32    UID;
        unsigned32    UserPerms;
        unsigned32    WriteCount;
};
typedef struct hpss_Attrs hpss_Attrs;
typedef hpss_Attrs hpss_Attrs_t;

```

### **Account**

Specifies opaque accounting information.

### **BitFileId**

Specifies the bitfile identifier.

### **RealmId**

Specifies the Realm identifier.

### **Comment**

Specifies the uninterpreted client supplied ASCII text.

### **CompositePerms**

Specifies the permission to an object after all ACLs have been examined and applied.

### **COSId**

Specifies the class of service ID.

### **DataLength**

Specifies the byte length of Data.

### **DMDataStateFlags**

A collection of three bits which describe the state of data which is stored in HPSS and simultaneously stored in some other filesystem such as XFS. The three bits which describe this state are:

CORE\_DMSTATE\_CACHE\_DATA\_VALID

CORE\_DMSTATE\_HPSS\_DATA\_VALID

CORE\_DMSTATE\_SYNC\_FILESET\_DATA

### **DMHandle**

Specifies a handle that points back to a DMAP managed object. This field is opaque data to the Core Server.

### **DMHandleLength**

Specifies the byte length of DMHandle.

### **DontPurge**

A flag indicating whether a file can be purged. If 1, the file cannot be purged.

### **EntryCount**

Specifies a read-only field which contains the number of entries contained in a directory. If the object is not

a directory, the value is not defined.

### **ExtendedACLs**

A flag that indicates whether an object has extended ACLs (ACL entries other than user\_obj, group\_obj, and other\_obj). If 1, the object has extended ACLs.

### **FamilyId**

Identifies the fileset family identifier.

### **FilesetHandle**

Specifies a Core Server object handle used to point to the root node of a fileset.

### **FilesetId**

Specifies the fileset identifier that uniquely identifies the fileset an object belongs to.

### **FilesetRootId**

Specifies a root ID of this fileset.

### **FilesetStateFlags**

Contains flag bits indicating the state of the fileset. The following constants define the possible states:

CORE_FS_STATE_READ	Read is permitted.
CORE_FS_STATE_WRITE	Write is permitted.
CORE_FS_STATE_DESTROYED	The fileset has been destroyed. Neither reading nor writing will be permitted
CORE_FS_STATE_READ_WRITE	A combination of READ and WRITE.
CORE_FS_STATE_COMBINED	A combination of all bit settings above.

### **FilesetType**

Specifies the type of the fileset the attributes are for. This is a read-only field. The following constants define the fileset types:

CORE_FS_TYPE_HPSS_ONLY	This fileset is an HPSS-only fileset.
CORE_FS_TYPE_ARCHIVED	This fileset is a backup copy of some other fileset.
CORE_FS_TYPE_NATIVE	This fileset is native to some other file system such as XFS.
CORE_FS_TYPE_MIRRORED	This fileset is a mirrored copy of some other fileset.

### **GID**

Specifies the principal group identifier.

### **GroupPerms**

Specifies the permissions granted to group members.

### **LinkCount**

Specifies the number of hard links to a file object.

**MACSecLabel**

A security label for the object.

**OpenCount**

Specifies the number of opens to a file object.

**OtherPerms**

Specifies the permissions granted to ‘other’ clients.

**ReadCount**

Specifies the number of times that all or part of the bitfile has been read.

**RegisterBitMap**

A bit vector used to indicate which fields in the attribute structure should trigger SSM notifications when changed.

**SetGIDBit**

For file objects:

0 = do not set GID to owner.

1 = set GID to owner.

**SetStickyBit**

For file objects:

0 = do not set the sticky bit.

1 = set the sticky bit.

**SetUIDBit**

For file objects:

0 = do not set UID to owner.

1= set UID to owner.

**SubSystemId**

Specifies the Subsystem ID.

**TimeCreated**

Specifies the time the object was created.

**TimeLastRead**

Specifies the last time the object was accessed.

**TimeLastWritten**

Specifies the last time the object was written.

**TimeModified**

Specifies the time the object was modified.

### **Type**

This field is not settable. Specifies the 'type' of the object:

NS_OBJECT_TYPE_DIRECTORY	directory
NS_OBJECT_TYPE_FILE	file
NS_OBJECT_TYPE_JUNCTION	junction
NS_OBJECT_TYPE_SYM_LINK	symbolic link
NS_OBJECT_TYPE_HARD_LINK	hard link

### **UID**

Specifies the User Identifier of the object's owner.

### **UserPerms**

Specifies the permissions granted to the owner of the object.

### **WriteCount**

Specifies the number of writes to a file object.

## **2.2.13. Name Service Object Attribute Bits - hpss\_AttrBits\_t**

### **Description**

Bits specifying the Name Service object attributes to retrieve or set.

### **Format**

The hpss\_AttrBits\_t has the following format:

```
typedef u_signed64                hpss_AttrBits_t;
```

## **2.2.14. Authentication Mechanism Type - hpss\_authn\_mech\_t**

### **Description**

This structure defines the different types of authentication mechanisms for HPSS.

### **Format**

The hpss\_authn\_mech\_t structure has the following format:

```
enum hpss_authn_mech_t {
    hpss_authn_mech_invalid = 0,
    hpss_authn_mech_krb5 = 1,
    hpss_authn_mech_unix = 2,
    hpss_authn_mech_gsi = 3,
    hpss_authn_mech_spkm = 4
};
typedef enum hpss_authn_mech_t hpss_authn_mech_t;
```



## 2.2.15. Authentication Token Type - hpss\_authz\_token\_t

### Description

This structure defines the different types of HPSS authentication types.

### Format

The hpss\_authz\_token\_t structure has the following format:

```
struct hpss_authz_token {
    byte Version;
    unsigned32 DataLength;
    byte Data[HPSS_AUTHZ_TOKEN_DATA_SZ];
    unsigned32 Timestamp;
    unsigned16 SourceAddType;
    unsigned32 SourceAddrLength;
    byte SourceAddr[16];
    unsigned32 CheckSumLength;
    byte CheckSum[HPSS_AUTHZ_TOKEN_CKSUM_SZ];
};
typedef struct hpss_authz_token hpss_authz_token;

typedef hpss_authz_token hpss_authz_token_t;
```

#### Version

The version number of the authentication token, currently at version 1.

#### DataLength

Length of token data in bytes. Maximum length is currently set at 128 bytes;

#### Data

The token data.

#### Timestamp

A timestamp obtained using the system time() subroutine. The time() subroutine returns the number of seconds since the Epoch (00:00:00 GMT, January 1, 1970).

#### SourceAddType

Type of the source address.

#### SourceAddrLength

Length of the source address.

#### SourceAddr

The source address.

#### CheckSumLength

Length in bytes of CheckSum. Maximum length is currently set at 64 bytes.

#### CheckSum

Checksum of the token data.

## 2.2.16. File Creation Hint Structure - `hpss_cos_hints_t`

### Description

The file creation hint structure contains information that allows clients to specify preferences or knowledge of file structure or access patterns that may affect operations of HPSS.

### Format

The COS hints has the following format:

```
struct hpss_cos_hints {
    unsigned32    COSId;
    char          COSName[HPSS_MAX_OBJECT_NAME];
    unsigned32    Flags;
    u_signed64    OptimumAccessSize;
    u_signed64    MinFileSize;
    u_signed64    MaxFileSize;
    unsigned32    AccessFrequency;
    unsigned32    TransferRate;
    unsigned32    AvgLatency;
    unsigned32    WriteOps;
    unsigned32    ReadOps;
    unsigned32    StageCode;
    unsigned32    StripeWidth;
    u_signed64    StripeLength;
};
typedef struct hpss_cos_hints hpss_cos_hints;
typedef hpss_cos_hints hpss_cos_hints_t;
```

#### **COSId**

The class of service that the bitfile belongs to.

#### **COSName**

Specifies the name of the class of service for this bitfile.

#### **Flags**

Flags used in bitfile creation, values are:

HINTS\_FORCE\_MAX\_SEGSZ - Force maximum segment size for the bitfile.

#### **OptimumAccessSize**

Specifies the block size in bytes for this class of service that yields the maximum data transfer rate.

#### **MinFileSize**

Specifies the minimum size in bytes of a bitfile in this class of service.

#### **MaxFileSize**

Specifies the maximum size in bytes to which the bitfile can grow and remain in this class of service.

#### **AccessFrequency**

Specifies the expected rate of access for the bitfile.

FREQ\_HOURLY

FREQ\_DAILY

FREQ\_WEEKLY

FREQ\_MONTHLY

FREQ\_ARCHIVE

### **TransferRate**

Specifies the approximate file transfer rate in kilobytes per second.

### **AvgLatency**

Specifies the time in seconds from when a request is received by a Core Server until data actually begins to transmit. This is typically non-zero for tape media.

### **WriteOps**

Specifies the valid write operations for the bitfile:

HPSS\_OP\_WRITE                      Allow write operations.

HPSS\_OP\_APPEND                    Allow append operations.

### **ReadOps**

Specifies the valid read operations for the bitfile:

HPSS\_OP\_READ                      Allow read operations.

### **StageCode**

Specifies the staging behavior desired:

COS\_STAGE\_NO\_STAGE                File is not to be staged on open. The data will be read from the current level in the hierarchy, or data may be explicitly staged by the client.

COS\_STAGE\_ON\_OPEN                Entire file is to be staged to the top level in the hierarchy before open returns.

COS\_STAGE\_ON\_OPEN\_ASYNC        Entire file is to be staged to the top level in the hierarchy without blocking in open. Reads / writes are blocked only until the portion of the file being accessed is staged.

COS\_STAGE\_ON\_OPEN\_BACKGROUND   File is to be staged in a background task.

### **StripeWidth**

Specifies the stripe width of the class of service.

### **StripeLength**

Specifies the stripe length of the class of service.

## 2.2.17. Class of Service Priorities - `hpss_cos_priorities_t`

### Description

The class of service priorities structure assists a client in selecting a COS for a bitfile. Structure use - dynamic memory tables.

### Format

The COS priorities has the following format:

```
struct hpss_cos_priorities {
    unsigned32 COSIdPriority;
    unsigned32 COSNamePriority;
    unsigned32 OptimumAccessSizePriority;
    unsigned32 MinFileSizePriority;
    unsigned32 MaxFileSizePriority;
    unsigned32 AccessFrequencyPriority;
    unsigned32 TransferRatePriority;
    unsigned32 AvgLatencyPriority;
    unsigned32 WriteOpsPriority;
    unsigned32 ReadOpsPriority;
    unsigned32 StageCodePriority;
    unsigned32 StripeWidthPriority;
    unsigned32 StripeLengthPriority;
};
typedef struct hpss_cos_priorities hpss_cos_priorities;
typedef hpss_cos_priorities hpss_cos_priorities_t;
```

Valid priority values are:

NO\_PRIORITY  
LOWEST\_PRIORITY  
LOW\_PRIORITY  
DESIRED\_PRIORITY  
HIGHLY\_DESIRED\_PRIORITY  
REQUIRED\_PRIORITY

#### **COSIdPriority**

Specifies the class of service ID priority for the class of service the bitfile should be in.

#### **COSNamePriority**

Specifies the class of service name priority for this bitfile.

#### **OptimumAccessSizePriority**

Specifies the priority for the block size for this class of service that yields the maximum data transfer rate.

#### **MinFileSizePriority**

Specifies the priority for the minimum size in bytes of a bitfile in this class of service.

### **MaxFileSizePriority**

Specifies the priority for the maximum size in bytes to which the bitfile can grow and remain in this class of service.

### **AccessFrequencyPriority**

Specifies the priority for the expected rate of access for the bitfile.

### **TransferRatePriority**

Specifies the priority for the class of service file transfer rate.

### **AvgLatencyPriority**

Specifies the class of service priority for the average latency time from request time until data begins to transfer.

### **WriteOpsPriority**

Specifies the priority for the valid write operations for the bitfile.

### **ReadOpsPriority**

Specifies the priority for the valid read operations for the bitfile.

### **StageCodePriority**

Specifies the priority for the desired stage code.

### **StripeWidthPriority**

Specifies the stripe width priority of the storage class.

### **StripeLengthPriority**

Specifies the stripe length priority of the storage class.

## **2.2.18. Class of Service Metadata Structure - hpss\_cos\_md\_t**

### **Description**

The Class of Service metadata structure contains information about the configuration of a Class of Service.

### **Format**

The Class of Service metadata structure has the following format:

```
struct hpss_cos_md {
    unsigned32  COSId;
    unsigned32  HierId;
    char        COSName[HPSS_MAX_OBJECT_NAME];
    unsigned32  OptimumAccessSize;
    unsigned32  Flags;
    u_signed64  MinFileSize;
    u_signed64  MaxFileSize;
    unsigned32  AccessFrequency;
    unsigned32  TransferRate;
    unsigned32  AvgLatency;
    unsigned32  WriteOps;
```

```

        unsigned32 ReadOps;
        unsigned32 StageCode;
};
typedef struct hpss_cos_md hpss_cos_md;
typedef hpss_cos_md hpss_cos_md_t;

```

### **COSId**

The class of service type. It indicates which of several classes of service the bitfile is in.

### **HierId**

The storage hierarchy associated with this Class of Service.

### **COSName**

Specifies the name of the class of service for this bitfile.

### **OptimumAccessSize**

Specifies the block size in bytes for this class of service that yields the maximum data transfer rate.

### **Flags**

Optionally specify one of the following options:

COS_ENFORCE_MAX_FILE_SIZE	If ON, bitfiles cannot be created in this COS with a size greater than MaxFileSize. Attempts to do so will result in the request being rejected with an error.
COS_FORCE_SELECTION	If ON, a client must explicitly select this COS in order to have a file assigned to it. If the client merely supplies general COS hints for a bitfile, this COS will not be selected.
COS_AUTO_STAGE_RETRY	If ON, HPSS will automatically retry a failed stage from the primary copy if a valid secondary copy exists.

### **MinFileSize**

Specifies the minimum size in bytes of a bitfile in this class of service.

### **MaxFileSize**

Specifies the maximum size in bytes to which the bitfile can grow and remain in this class of service.

### **AccessFrequency**

Specifies the expected rate of access for the bitfile.

FREQ\_HOURLY

FREQ\_DAILY

FREQ\_WEEKLY

FREQ\_MONTHLY

FREQ\_ARCHIVE

### **TransferRate**

Specifies the approximate file transfer rate in kilobytes per second.

#### **AvgLatency**

Specifies the time in seconds from when a request is received by a Core Server until data actually begins to transmit. This is typically non-zero for tape media.

#### **WriteOps**

Specifies the valid write operations for the bitfile:

HPSS\_OP\_WRITE            Allow write operations.

HPSS\_OP\_APPEND         Allow append operations.

#### **ReadOps**

Specifies the valid read operations for the bitfile:

HPSS\_OP\_READ           Allow read operations.

#### **StageCode**

Specifies the staging behavior desired:

COS\_STAGE\_NO\_STAGE            File is not to be staged on open. The data will be read from the current level in the hierarchy, or data may be explicitly staged by the client.

COS\_STAGE\_ON\_OPEN            Entire file is to be staged to the top level in the hierarchy before open returns.

COS\_STAGE\_ON\_OPEN\_ASYNC      Entire file is to be staged to the top level in the hierarchy without blocking in open. Reads/writes are blocked only until the portion of the file being accessed is staged.

COS\_STAGE\_ON\_OPEN\_BACKGROUND   File is to be staged as a background task.

## **2.2.19. HPSS Directory Entry - hpss\_dirent\_t**

### **Description**

The HPSS directory entry structure contains the directory's name and a handle to the Core Server for this directory. In addition, it contains the offset of the next directory entry.

### **Format**

```
typedef struct hpss_dirent {
    u_signed64      d_offset;
    ns_ObjHandle_t  d_handle;
    unsigned16      d_reclen;
    unsigned16      d_namelen;
    char            d_name[HPSS_MAX_FILE_NAME];
} hpss_dirent_t;
```

#### **d\_offset**

The offset of the next directory entry.

**d\_handle**

The handle to the Core Server for the directory.

**d\_reclen**

The record length of the directory.

**d\_namelen**

The number of characters in the directory name.

**d\_name**

The name of the directory.

## 2.2.20. Extended Transaction Outcome - hpss\_ExTrans\_Outcome\_t

### Description

Indicates the outcome of an extended transaction.

### Format

This data type is declared as follows:

```
typedef unsigned32 hpss_ExTransOutcome_t;
```

## 2.2.21. File Attribute Structure - hpss\_fileattr\_t

### Description

The file attribute structure contains file attributes that are managed by the Core Server.

### Format

The file attribute structure has the following format:

```
typedef struct hpss_fileattr {  
    ns_ObjHandle_t    ObjectHandle;  
    hpss_Attrs_t      Attrs;  
} hpss_fileattr_t;
```

**ObjectHandle**

Specifies the handle that refers to the open file or directory.

**Attrs**

Specifies the file attributes managed by the Core Server.

## 2.2.22. File Attributes Bit Data Type - hpss\_fileattrbits\_t

### Description

This data type specifies the HPSS File Attributes Bits.

### Format

This data type is declared as follows:



```
typedef u_signed64 hpss_AttrBits_t;
typedef hpss_AttrBits_t hpss_fileattrbits_t;
```

## 2.2.23. Extended File Attribute Structure - hpss\_xfileattr\_t

### Description

The file attribute structure contains file attributes that are managed by the Core Server. This structure allows the user to retrieve the extended attributes.

### Format

The file attribute structure has the following format:

```
struct hpss_xfileattr {
    ns_ObjHandle_t  ObjectHandle;
    hpss_Attrs_t  Attrs;
    bf_sc_attr_t  SCAttrib[HPSS_MAX_STORAGE_LEVELS];
};
typedef struct hpss_xfileattr hpss_xfileattr;
typedef hpss_xfileattr hpss_xfileattr_t;
```

#### ObjectHandle

Specifies the handle that refers to the open file or directory.

#### Attrs

Specifies the file attributes.

#### SCAttrib

Specifies the storage class attributes for each valid level in the hierarchy.

## 2.2.24. Global Fileset Entry Structure - hpss\_global\_fsent\_t

### Description

The global fileset entry structure contains global fileset information.

### Format

The global fileset entry structure has the following format:

```
typedef struct {
    u_signed64      FilesetId;
    unsigned char   FilesetName[HPSS_MAX_FS_NAME];
    uuid_t          GatewayUUID;
    uuid_t          CoreServerUUID;
} hpss_global_fsent_t;
```

#### FilesetId

The unique fileset identifier.

#### FilesetName

The unique name of the fileset.

**GatewayUUID**

The identifier of the DMAP gateway that manages the fileset.

**CoreServerUUID**

The identifier of the Core Server that manages the fileset.

## 2.2.25. Parallel I/O Callback Type - **hpss\_pio\_cb\_t**

### Description

This HPSS data type declares a callback function.

### Format

The format for this data type is:

```
typedef int (*hpss_pio_cb_t) (
    void *,          /* IN - Call back argument */
    u_signed64,      /* IN - Local file offset */
    unsigned int *,  /* IN/OUT - Length of requested data */
    void **);        /* IN/OUT - Pointer to first byte of data */
```

## 2.2.26. Parallel I/O Gap Information Type - **hpss\_pio\_gapinfo\_t**

### Description

The **hpss\_pio\_gapinfo\_t** data type provides the required information determining a gap in the data.

### Format

The format for this data type is:

```
typedef struct hpss_pio_gapinfo_s {
    u_signed64      Offset;
    u_signed64      Length;
} hpss_pio_gapinfo_t;
```

**Offset**

Starting offset of the gap in bytes.

**Length**

Length of the data gap in bytes.

## 2.2.27. Parallel Stripe Group Type - **hpss\_pio\_grp\_t**

### Description

This data type is a pointer to a parallel I/O stripe group.

### Format

The format of this data type is:

```
typedef void *hpss_pio_grp_t;
```

## 2.2.28. HPSS Parallel I/O Operation Type - hpss\_pio\_operation\_t

### Description

This data type defines the parallel I/O operations.

### Format

The format of this data type is:

```
typedef enum { HPSS_PIO_READ, HPSS_PIO_WRITE } hpss_pio_operation_t;
```

## 2.2.29. HPSS Parallel I/O Parameters - hpss\_pio\_params\_t;

### Description

This data structure defines the parallel I/O parameters.

### Format

The format of the hpss\_pio\_params\_t data structure is:

```
typedef struct hpss_pio_params_s {  
    hpss_pio_operation_t Operation;  
    unsigned32          ClnStripeWidth;  
    unsigned32          BlockSize;  
    unsigned32          FileStripeWidth;  
    unsigned32          IOTimeOutSecs;  
    hpss_pio_transport_t Transport;  
    hpss_pio_options_t   Options;  
} hpss_pio_params_t;
```

#### Operation

Specifies the type of operation. Valid values are:

HPSS\_PIO\_READ

HPSS\_PIO\_WRITE

#### ClnStripeWidth

Number of data elements in the client stripe.

#### BlockSize

Number of bytes in each data element in the data stripe.

#### FileStripeWidth

Number of data elements in the file stripe.

#### IOTimeOutSecs

Number of seconds to wait for I/O. Zero will get the default of 30 minutes.

#### Transport

Type of data transport. Valid values are:

HPSS\_PIO\_TCPIP

**Options**

Parallel I/O transport options. Valid values are:

HPSS\_PIO\_PUSH

HPSS\_PIO\_HANDLE\_GAP

## 2.2.30. HPSS Parallel I/O Transport Type - `hpss_pio_transport_t`

### Description

This data type defines the types of parallel I/O transport types available.

### Format

The format of this data type is:

```
typedef enum { HPSS_PIO_TCPIP } hpss_pio_transport_t;
```

## 2.2.31. HPSS I/O Request ID Type - `hpss_reqid_t`

### Description

The `hpss_reqid_t` data type defines a unique HPSS I/O request identifier.

### Format

The format for this data type is:

```
typedef unsigned32 hpss_reqid_t;
```

## 2.2.32. HPSS RPC Protection Level Type - `hpss_rpc_prot_level_t`

### Description

This data type defines the level of protection to be used for the I/O transaction.

### Format

The format of this data type is:

```
enum hpss_rpc_prot_level_t {  
    hpss_rpc_protect_connect = 1,  
    hpss_rpc_protect_pkt = 2,  
    hpss_rpc_protect_pkt_integ = 3,  
    hpss_rpc_protect_pkt_privacy = 4  
};  
typedef enum hpss_rpc_prot_level_t hpss_rpc_prot_level_t;
```

## 2.2.33. HPSS File Statistics Data Structure - `hpss_stat_t`

### Description

This structure contains the variables specifying the file statistics.

## Format

The format of the `hpss_stat_t` structure is:

```
typedef struct hpss_stat {
    unsigned32      st_dev;
    u_signed64      st_ino;
    unsigned16      st_nlink;
    unsigned16      st_flag;
    unsigned32      st_uid;
    unsigned32      st_gid;
    unsigned32      st_rdev;
    u_signed64      st_ssize;
    timestamp_sec_t hpss_st_atime;
    signed32        st_atime_n;
    timestamp_sec_t hpss_st_mtime;
    signed32        st_mtime_n;
    timestamp_sec_t hpss_st_ctime;
    signed32        st_ctime_n;
    unsigned32      st_blksize;
    unsigned32      st_blocks;
    signed32        st_vfstype;
    unsigned32      st_vfs;
    unsigned32      st_type;
    unsigned32      st_gen;
    u_signed64      st_size;
    unsigned32      st_mode;
} hpss_stat_t;
```

### *st\_dev*

Identifier of the device containing a directory entry for an HPSS file.

### *st\_ino*

File serial number.

### *st\_nlink*

Number of links to the file.

### *st\_flag*

Flag word.

### *st\_uid*

User ID of the file's owner.

### *st\_gid*

Group ID of the file's group.

### *st\_rdev*

Identifier of a character or block special file only.

### *st\_ssize*

64 bit file size.

**hpss\_st\_atime**

Time of last access.

**st\_atime\_n**

Not used.

**hpss\_st\_mtime**

Time of last modification.

**st\_mtime\_n**

Not used.

**hpss\_st\_ctime**

Time of last file status change.

**st\_ctime\_n**

Not used.

**st\_blksize**

Optimal blocksize for file system I/O operations.

**st\_blocks**

Actual number of blocks allocated.

**st\_vfstype**

Type of virtual file system. Not used.

**st\_vfs**

VFS number. Not used.

**st\_type**

Vnode type. Not used.

**st\_gen**

Inode generation number. Not used.

**st\_size**

64 bit file size.

**st\_mode**

File mode. Valid POSIX values for this are:

<u>S_IFMT</u>	Mask used to determine file type.
S_IFREG	Regular file
S_IFDIR	Directory

S_IFBLK	Block special file.
S_IFCHR	Character special file.
S_IFIFO	First in first out.
S_ISUID	Set user id on execution.
S_ISGID	Set group id on execution.
S_IRWXU	Read, write, execute permissions for owner.
S_IRUSR	Read permission for owner.
S_IWUSR	Write permission for owner.
S_IXUSR	Execute, search permission for owner.
S_IRWXG	Read, write, execute permissions for group.
S_IRGRP	Read permission for group.
S_IWGRP	Write permission for group.
S_IXGRP	Execute, search permissions for group.
S_IRWXO	Read, write, execute permissions for others.
S_IROTH	Read permission for others.
S_IWOTH	Write permission for others.
S_IXOTH	Execute, search permissions for others.

## 2.2.34. HPSS VFS File Statistics Structure - `hpss_statvfs_t`

### Description

This data structure contains the HPSS vfs file statistics.

### Format

The `hpss_statvfs_t` structure's format is:

```
typedef struct hpss_statvfs {
    unsigned32 f_bsize;
    unsigned32 f_blocks;
    unsigned32 f_bfree;
    unsigned32 f_bavail;
    unsigned32 f_files;
    unsigned32 f_ffree;
    unsigned32 f_fsid;
    unsigned32 f_namemax;
    char      f_fstr[STATVFS_NAME_LENGTH];
    char      f_fpack[STATVFS_NAME_LENGTH];
} hpss_statvfs_t;
```

#### *f\_bsize*

Preferred file system block size.

**f\_blocks**

Total data blocks in file system.

**f\_bfree**

Number of free blocks in file system.

**f\_bavail**

Number of free blocks available to non-superuser.

**f\_files**

Total number of file nodes (inode in JFS).

**f\_ffree**

Number of free file nodes in file system.

**f\_fsid**

The file system's identifier.

**f\_namemax**

The maximum filename length.

**f\_str**

A file system specific string.

**f\_pack**

File system pack name.

## 2.2.35. HPSS Name Space Attributes Structure - hpss\_vattr\_t

### Description

This structure contains the attributes of an HPSS name space object.

### Format

The format for this structure is:

```
typedef struct hpss_vattr {
    hpss_type_t      va_type;
    hpss_mode_t      va_mode;
    hpss_uid_t       va_uid;
    hpss_gid_t       va_gid;
    unsigned32       va_fsid;
    unsigned32       va_serialno;
    unsigned32       va_nlink;
    unsigned32       va_rdev;
    unsigned32       va_nid;
    unsigned32       va_chan;
    unsigned32       va_aclsize;
    u_signed64       va_size;
    u_signed64       va_prefiosize;
```



u_signed64	va_blocksize;
u_signed64	va_numblocks;
u_signed64	va_ftid;
timestamp_sec_t	va_atime;
timestamp_sec_t	va_mtime;
timestamp_sec_t	va_ctime;
unsigned char	*va_acl;
unsigned32	va_realm;
ns_ObjHandle_t	va_objhandle;
hpsoid_t	va_soid;
cos_t	va_cos;
acct_rec_t	va_account;
} hpss_vattr_t;	

#### **va\_type**

HPSS object type.

#### **va\_mode**

Unix permissions of object.

#### **va\_uid**

Unix id of object owner.

#### **va\_gid**

Unix group id of object owner.

#### **va\_fsid**

Not used.

#### **va\_serialno**

Not used.

#### **va\_nlink**

Number of links to this object.

#### **va\_rdev**

Not used.

#### **va\_nid**

Network id for this object.

#### **va\_chan**

Channel for object device.

#### **va\_aclsize**

ACL size in bytes.

#### **va\_size**

HPSS object size in bytes.

**va\_prefiosize**

Preferred I/O size.

**va\_blocksize**

HPSS object block size.

**va\_numblocks**

HPSS object block count.

**va\_fid**

Fileset id.

**va\_atime**

Time of last access.

**va\_mtime**

Last modification time.

**va\_ctime**

Time of last status change.

**va\_acl**

Pointer to Access Control List.

**va\_realm**

Security realm id.

**va\_objhandle**

Name Server object handle.

**va\_soid**

HPSS bitfile identifier.

**va\_cos**

HPSS class of service identifier.

**va\_acct**

HPSS account index.

## 2.2.36. HPSS UUID Type - hpss\_uuid\_t

### Description

This structure defines an HPSS UUID.

### Format

The structure's format is:

```
struct hpss_uuid {
```

```

        unsigned32 time_low;
        unsigned16 time_mid;
        unsigned16 time_hi_and_version;
        unsigned8 clock_seq_hi_and_reserved;
        unsigned8 clock_seq_low;
        byte node[6];
    };
    typedef struct hpss_uuid hpss_uuid;
    typedef hpss_uuid hpss_uuid_t;

```

## 2.2.37. HPSS Namespec Type - namespec\_type\_t

### Description

This data type defines the types of namespace translation.

### Format

The format for this data type is:

```

typedef enum {
    NAMESPEC_SKIP,
    NAMESPEC_REALM,
    NAMESPEC_USER,
    NAMESPEC_GROUP
} namespec_type_t;

```

## 2.2.38. Name Service ACL Conformant Array - ns\_ACLConfArray\_t

### Description

The ns\_ACLConfArray\_t structure describes a template for a conformant array of Name Service ACL entries.

### Format

The ns\_ACLConfArray\_t structure has the following format:

```

typedef struct {
    signed32      Length;
    ns_ACLEntry_t ACLEntry[*];
} ns_ACLConfArray_t;

```

#### Length

Specifies the number of ACL entries in the array.

#### ACLEntry

The array of ACL entries.

## 2.2.39. Name Service Access Control List Entry - ns\_ACLEntry\_t

### Description

The ns\_ACLEntry\_t structure describes a Name Service ACL entry. Each entry contains information such

as the type of entry (i.e., for a group or individual user), the identity and location of the user or group and the permissions that are allowed.

#### Format

The `ns_ACLEntry_t` structure has the following format:

```
struct ns_ACLEntry {
    u_char      EntryType;
    u_char      Perms;
    unsigned32  EntryId;
    unsigned32  RealmId;
};
typedef struct ns_ACLEntry ns_ACLEntry;
typedef ns_ACLEntry ns_ACLEntry_t;
```

##### EntryType

Identifies the type of this ACL entry. These correspond to the following ACL tag types: `user_obj`, `user_obj_delegate`, `user`, `user_delegate`, `foreign user`, `foreign_user_delegate`, `group_obj`, `group_obj_delegate`, `group`, `group_delegate`, `foreign_group`, `foreign_group_delegate`, `other_obj`, `other_obj_delegate`, `foreign_other`, `foreign_other_delegate`, `any_other`, `any_other_delegate`, `mask_obj`, or `unauthenticated`.

##### Perms

Specifies the permissions or access rights.

##### EntryId

Depending on the `EntryType`, it can specify an identifier (usually a UID or GID).

##### RealmId

Specifies the realm identifier.

## 2.2.40. Name Service Directory Entry - `ns_DirEntry_t`

#### Description

The Name Service directory entry structure defines the contents of an HPSS directory entry.

#### Format

The Name Service directory entry structure has the following format:

```
struct DirEntryTag {
    char      Name[HPSS_MAX_FILE_NAME];
    ns_ObjHandle_t  ObjHandle;
    u_signed64  ObjOffset;
    hpss_Attrs_t  Attrs;
};
typedef struct DirEntryTag DirEntryTag;
typedef DirEntryTag ns_DirEntry_t;
```

##### Name

Specifies the name of the directory entry.

### ObjHandle

Specifies the Name Service object handle of the directory entry.

### ObjOffset

Specifies the offset of the entry within the directory.

### Attrs

Specifies attributes of the directory entry.

## 2.2.41. Name Service Fileset Attributes Structure - ns\_FilesetAttrs\_t

### Description

The Name Service fileset attribute structure contains fields for the various attributes (metadata) that the Core Server maintains for a fileset.

### Format

The Name Server fileset attributes structure has the following format:

```
struct ns_FilesetAttrs {
    u_signed64      RegisterBitMap;
    u_signed64      ChangedRegisterBitMap;
    unsigned32      ClassOfService;
    unsigned32      FamilyId;
    ns_ObjHandle_t  FilesetHandle;
    u_signed64      FilesetId;
    char            FilesetName[HPSS_MAX_FS_NAME_LENGTH];
    unsigned32      FilesetType;
    hpss_uuid_t     GatewayUUID;
    unsigned32      StateFlags;
    unsigned32      SubSystemId;
    byte            UserData[NS_FS_MAX_USER_DATA];
    u_signed64      DirectoryCount;
    u_signed64      FileCount;
    u_signed64      HardLinkCount;
    u_signed64      JunctionCount;
    u_signed64      SymLinkCount;
};
typedef struct ns_FilesetAttrs ns_FilesetAttrs;
typedef ns_FilesetAttrs ns_FilesetAttrs_t;
```

### RegisterBitMap

A bit vector where each bit corresponds to a field in the record.

### ChangedRegisterBitMap

A bit vector where each bit corresponds to a changed field in the record.

### ClassOfService

The COS service configured for this fileset.

### FamilyId

The fileset family identifier. This id is opaque to the Core Server.

### **FilesetHandle**

A Core Server object handle which points to the root node of the fileset.

### **FilesetId**

The unique identifier of this fileset.

### **FilesetName**

The unique human readable fileset name.

### **FilesetType**

Specifies the type of the fileset the attributes are for. This is a read-only field. The following constants define the fileset types:

CORE_FS_TYPE_HPSS_ONLY	This fileset is an HPSS-only fileset.
CORE_FS_TYPE_ARCHIVED	This fileset is a backup copy of some other fileset.
CORE_FS_TYPE_MIRRORED	This fileset is a mirrored copy of some other fileset such as an XFS fileset.

### **GatewayUUID**

The identifier of the gateway that processes DMAP requests for the fileset.

### **StateFlags**

The flags that defined the state of the fileset. Valid values include:

CORE_FS_STATE_READ	The fileset allows reads.
CORE_FS_STATE_WRITE	The fileset allows writes.
CORE_FS_STATE_READ_WRITE	The fileset allows reads and writes.
CORE_FS_STATE_DESTROYED	The fileset allows no access.

### **SubSystemId**

The HPSS storage subsystem that this name space object belongs to.

### **UserData**

Uninterpreted data supplied by the client. This data can be ASCII, binary, or both.

### **DirectoryCount**

The current number of directories in the fileset.

### **FileCount**

The current number of files in the fileset.

### **HardLinkCount**

The current number of hard links in the fileset.

### **JunctionCount**

The current number of junctions in the fileset.

**SymLinkCount**

The current number of symbolic links in the fileset.

## 2.2.42. Name Service Fileset Attribute Bits - ns\_FilesetAttrBits\_t

### Description

Bits specifying the Name Service fileset attribute bits to retrieve or set.

### Format

The ns\_FilesetAttrBits\_t has the following format:

```
typedef u_signed64 ns_FilesetAttrBits_t;
```

## 2.2.43. Name Service Object Handle Structure - ns\_ObjHandle\_t

### Description

The Name Service object handle structure contains information that allows the Core Server to identify the metadata record for an object.

### Format

The Name Service object handle structure has the following format:

```
struct ns_ObjHandle {
    u_signed64  ObjId;
    u_signed64  FileId;
    byte        Type;
    byte        Flags;
    unsigned16  Generation;
    hpss_uuid_t CoreServerUUID;
};
typedef struct ns_ObjHandle ns_ObjHandle;
typedef ns_ObjHandle ns_ObjHandle_t;
```

**ObjId**

Specifies a unique Core Server object identifier, the Relative Sequence Number (RSN) of the record containing the metadata for the object.

**FileId**

If the Type field specifies a hardlink this is the RSN of the record containing the metadata for the original file. For all other Types this field is equal to the ObjId.

**Type**

Specifies the 'type' of the object: file, directory, junction, symbolic link, or hard link.

**Flags**

Specifies a bit vector whose bits convey additional information about the object handle. The defined bit positions for the Flags field are:

NS\_OH\_FLAG\_FILESET\_ROOT                      Handle is for the root node of a fileset.

**Generation**

Specifies a random number used to detect stale object handles.

**CoreServerUUID**

Specifies the UUID of the Core Server that issued this object handle.

## 2.2.44. HPSS Object Handle Structure - `hpss_object_handle_t`

### Description

This structure contains information pertaining to an HPSS object.

### Format

The `hpss_object_handle_t` has the following format:

```
struct hpss_object_handle {
    signed32 ObjectPtr;
    hpss_uuid_t ObjectID;
    signed32 ObjectType;
    signed32 ConnectionType;
    signed32 ClientPtr;
};
typedef struct hpss_object_handle hpss_object_handle;
typedef hpss_object_handle hpss_object_handle_t;
```

**ObjectPtr**

Pointer to an HPSS object.

**ObjectID**

Unique identifier for the object.

**ObjectType**

The object type:

CONNECTION\_OBJECT

SESSION\_OBJECT

**ConnectionType**

Connection type of the object.

**ClientPtr**

Pointer to the client.

## 2.2.45. Purge Lock Flag - `purgelock_flag_t`

### Description

Flag specifying whether a file should have its purgelock status set or cleared.



## Format

The purgelock\_flag\_t structure has the following format:

```
typedef enum {
    PURGE_UNLOCK = 0,    /* purge unlock the file */
    PURGE_LOCK      /* purge lock the file */
} purgelock_flag_t;
```

## 2.2.46. Core Server Physical Volume Attributes - pv\_list\_element\_t

### Description

This structure contains physical volume location information for specified physical volume.

### Format

The Core Server physical volume attributes have the following format:

```
typedef struct pv_list_element {
    char Name[HPSS_PV_NAME_SIZE];
    unsigned32 Flags;
} pv_list_element_t;
```

#### Name

Specifies the physical volume name.

#### Flags

Specifies the location of the physical volume. The following constants may be set in this field:

PV\_MOUNT\_SUCCESS

PV\_UNMOUNT\_NOENT

PV\_UNMOUNT\_SUCCESS

PV\_ON\_SHELF

## 2.2.47. Core Server Physical Volume Attributes Conformant Array - pv\_list\_t

### Description

The pv\_list\_t structure describes a template for a conformant array of Core Server Physical Volume Attribute elements.

### Format

The Core Server physical volume attribute conformant array has the following format:

```
struct pv_list {
    struct {
        u_int List_len;
        pv_list_element_t *List_val;
    } List;
};
```

```
typedef struct pv_list pv_list;
typedef pv_list pv_list_t;
```

### List\_len

Specifies the number of physical volume attribute elements in the array.

### List\_val

A conformant array of physical volume attribute elements.

## 2.2.48. HPSS Security User Credentials - sec\_cred\_t

### Description

The HPSS Security User Credentials structure contains information about the user credentials. This information can only be obtained by an authorized client using HPSS security mechanisms.

### Format

The format of the sec\_cred\_t structure is:

```
struct sec_cred {
    char          Name[HPSS_MAX_USER_NAME] ;
    char          RealmName[HPSS_MAX_REALM_NAME] ;
    char          Directory[HPSS_MAX_PATH_NAME] ;
    char          UserShell[HPSS_MAX_USER_SHELL] ;
    unsigned32    RealmId;
    unsigned32    Uid;
    unsigned32    Gid;
    hpss_uuid_t   Uuid;
    acct_rec_t    DefAccount;
    acct_rec_t    CurAccount;
    unsigned32    NumGroups;
    unsigned32    AltGroups[HPSS_NGROUPS_MAX] ;
};
typedef struct sec_cred sec_cred;
typedef sec_cred sec_cred_t;
```

### Name

User name.

### RealmName

Realm name where principal resides.

### Directory

Directory name.

### UserShell

User's shell.

### RealmId

Identifier of realm where principal resides.

**Uid**

Unix user identifier.

**Gid**

Unix group identifier.

**Uuid**

Unique user identifier.

**DefAccount**

The accounting code that is used when a current account code has not been specified.

**CurAccount**

When a current accounting code is specified, this code is applied to new files or directories.

**NumGroups**

The number of groups to which this principal is a member.

**AltGroups**

An array of groups to which this principal is a member.

## 2.2.49. Subsystem Statistics - **subsys\_stats\_t**

### **Description**

This structure contains statistical information for a subsystem including counts of stages, migrates, purges, and deletions. In addition, the structure includes a timestamp indicating when the counts began.

### **Format**

The subsystem statistics have the following format:

```
typedef struct subsys_stats {  
    unsigned32      StageCount;  
    unsigned32      MigrationCount;  
    unsigned32      PurgeCount;  
    unsigned32      DeleteCount;  
    timestamp_sec_t  TimeLastReset;  
} subsys_stats_t;
```

**StageCount**

Specifies the number of stages which have occurred since the last reset.

**MigrationCount**

Specifies the number of migrations which have occurred since the last reset.

**PurgeCount**

Specifies the number of purges which have occurred since the last reset.

**DeleteCount**

Specifies the number of deletes which have occurred since the last reset.

**TimeLastReset**

Specifies the time of the last reset (all counts were set to 0).

## 2.2.50. Timestamp Seconds Type - timestamp\_sec\_t

### Description

This data type specifies a timestamp in seconds.

### Format

The format for this data type is:

```
typedef unsigned32 timestamp_sec_t;
```

## 2.3. Network Options

### 2.3.1. Network Options Functions

#### 2.3.1.1. netopt\_FindEntry

##### Purpose

Returns an HPSS network option entry for the specified IP address.

##### Synopsis

```
#include "hpss_netopt.h"
int
netopt_FindEntry(
    unsigned int      NetAddr,          /* IN */
    netopt_entry_t    **RetTableEntryPtr; /* OUT */
)
```

##### Description

The **netopt\_FindEntry** function searches the HPSS network option entries for an entry that matches the IP address specified by *NetAddr*, and returns a pointer to the entry in the value pointed to by *RetTableEntryPtr*.

##### Parameters

<i>NetAddr</i>	The IP address of the machine or network for which a match is requested in the HPSS network options table.
<i>RetTableEntryPtr</i>	Pointer to the area where the HPSS network option entry pointer will be returned.

##### Return Values

If an HPSS network option entry that corresponds to the specified IP address is found, then a value of zero is returned and a pointer to the entry is returned in the area specified by *RetTableEntryPtr*. Otherwise, a negative value is returned which describes the error, as defined below.

##### Error Conditions

HPSS_ENOMEM	Could not allocate memory for the network options table.
HPSS_ENOENT	The HPSS network options configuration file does not exist or no entry could be found that corresponds to the specified IP address.
HPSS_ESYSTEM	An operating system service failed.

#### See Also

**netopt\_GetWriteSize.**

#### Notes

None

### 2.3.1.2. netopt\_GetWriteSize

#### Purpose

Returns the size in bytes to be used for writes to the TCP/IP connection referred to by either a socket descriptor or IP address.

#### Synopsis

```
#include "hpss_netopt.h"
int
netopt_GetWriteSize(
    int                SocketDescriptor,      /* IN */
    unsigned int       IpAddr);               /* IN */
```

#### Description

The **netopt\_GetWriteSize** function returns the configured size, in bytes, to be used for writing data to the connection referred to by either *SocketDescriptor* or *NetAddr*. The function searches the HPSS network option entries for an entry that matches the IP address specified by *NetAddr*, if non-zero and otherwise based on the local address corresponding to the socket referred to by *SocketDescriptor*. If an entry is found and contains a non-zero network write size, that value is returned; otherwise the environment variable **HPSS\_TCP\_WRITESIZE** is interrogated - if it is set and contains a non-zero value, that value is returned, otherwise zero is returned.

#### Parameters

<i>SocketDescriptor</i>	File descriptor referring to the open TCP/IP connection over which the request is to be sent.
<i>IpAddr</i>	The IP address of the machine or network for which a match is requested in the HPSS network options table.

#### Return Values

The size, in bytes, that is configured for the specified network address is returned if any values have been specified. Otherwise the default value is returned.

#### Error Conditions

None.

#### See Also

**netopt\_FindEntry.**

**Notes**

None.

## 2.3.2. Network Options Data Definitions

### 2.3.2.1. Network Options Entry - netopt\_entry\_t

**Description**

The Network Options Entry structure contains the configuration information for a particular network (IP) address. This information is used to allow different networking options to be utilized for different HPSS Mover and/or HPSS client machines and networks

**Format**

The Network Options Entry structure has the following format:

```
typedef struct netopt_entry {  
    unsigned long    IPAddr;  
    unsigned long    NetMask;  
    unsigned long    RFC1323;  
    unsigned long    SendSpace;  
    unsigned long    RecvSpace;  
    unsigned long    WriteSize;  
    unsigned long    TCPNodelay;  
    unsigned long    Reserved2;  
} netopt_entry_t;
```

**IpAddr**

The IP address for this entry.

**NetMask**

The network mask to be applied to the incoming address to determine whether this entry applies to that address.

**RFC1323**

Indicates whether RFC 1323 (large TCP window size support) should be enabled for this address. A value of zero indicates that RFC 1323 support should be disabled; a non-zero value indicates that RFC 1323 support should be enabled.

**SendSpace**

The value to be used for the socket send buffer size.

**RecvSpace**

The value to be used for the socket receive buffer size.

**WriteSize**

The maximum number of bytes that should be written to a network connection corresponding to this entry with a single I/O request.

### **TCPNodelay**

Indicates whether the algorithm employed to try and coalesce small writes to a TCP connection should be disabled. A value of zero indicates that the algorithm should not be disabled; a non-zero value indicates that the algorithm should be disabled.

### **Reserved2**

This field is currently unused.





## Chapter 3. Math Library

This chapter specifies the HPSS 64-bit arithmetic programming interface. Specifically, the following information is provided:

- Application Programming Interfaces (APIs)
- Data Definitions

### API Interfaces

This section describes all API interfaces which are provided for use by another HPSS subsystem or by a client external to HPSS. The API interface specification includes the following information:

- Name
- Synopsis
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Notes

#### 3.1.1. **add64m**

##### Purpose

Add two unsigned 64-bit integers.

##### Synopsis

```
#include "u_signed64.h"

u_signed64 add64m(
    u_signed64  ll1, /* IN */
    u_signed64  ll2); /* IN */
```

##### Description

The *add64m* macro is called to add two unsigned 64-bit integers.

##### Parameters

<i>ll1</i>	Specifies the first unsigned 64-bit integer.
<i>ll2</i>	Specifies the second unsigned 64-bit integer.

##### Return Values

Upon completion, an unsigned 64-bit integer is returned.

##### Error Conditions

None.

## See Also

**add64\_3m.**

## Notes

None.

## 3.1.2. add64\_3m

### Purpose

Add two unsigned 64-bit integers and store the result in a separate integer field.

### Synopsis

```
#include "u_signed64.h"

void add64_3m(
    u_signed64 ll1, /* OUT */
    u_signed64 ll2, /* IN */
    u_signed64 ll3); /* IN */
```

### Description

The **add64\_3m** macro is called to add two unsigned 64-bit integers and store the result into a third unsigned 64-bit integer.

### Parameters

<i>ll1</i>	Specifies the result of the addition of the 2 unsigned 64-bit operands.
<i>ll2</i>	Specifies the first unsigned 64-bit integer addition operand.
<i>ll3</i>	Specifies an unsigned 64-bit integer to add to the first operand.

### Return Values

None. The result of the addition is stored in the first parameter.

### Error Conditions

None.

## See Also

**add64m.**

## Notes

None.

## 3.1.3. and64m

### Purpose

Find the bitwise AND of two unsigned 64-bit values.

### Synopsis

```
#include "u_signed64.h"

u_signed64 and64m (
    u_signed64  l11, /* IN */
    u_signed64  l12); /* IN */
```

### Description

The *and64m* macro is called to perform an AND operation of two unsigned 64-bit integer values.

### Parameters

<i>l11</i>	Specifies the first unsigned 64-bit integer.
<i>l12</i>	Specifies the second unsigned 64-bit integer.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

### Error Conditions

None.

### See Also

*not64m*, or *or64m*.

### Notes

None.

## 3.1.4. bld64m

### Purpose

Build an unsigned 64-bit integer from two unsigned 32-bit integers.

### Synopsis

```
#include "u_signed64.h"

u_signed64 bld64m(
    unsigned32  l1, /* IN */
    unsigned32  l2); /* IN */
```

### Description

The *bld64m* macro is called to construct an unsigned 64-bit integer from 2 unsigned 32-bit integers.

### Parameters

<i>l1</i>	Specifies the unsigned 32-bit integer which will occupy the high order portion of the unsigned 64 -bit integer.
<i>l2</i>	Specifies the unsigned 32-bit integer which will occupy the low order portion of the unsigned 64 -bit integer.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**cast64m.**

#### Notes

none.

### 3.1.5. cast64m

#### Purpose

Cast an unsigned 32-bit integer into an unsigned 64-bit integer.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 cast64m(
    unsigned32 11); /* IN */
```

#### Description

The **cast64m** macro is called to cast an unsigned 32-bit integer into an unsigned 64-bit integer.

#### Parameters

*11* Specifies an unsigned 32-bit integer to be cast into an unsigned 64-bit integer.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**cast32m.**

#### Notes

None.

### 3.1.6. div64m

#### Purpose

Divide an unsigned 64-bit integer by an unsigned 32-bit integer.

#### Synopsis

```
#include "u_signed64.h"
```

```

u_signed64 div64m(
    u_signed64  l11,  /* IN */
    unsigned32  l1); /* IN */

```

### Description

The *div64m* macro is called to divide an unsigned 64-bit value by an unsigned 32-bit value.

### Parameters

<i>l11</i>	Specifies an unsigned 64-bit integer numerator.
<i>l1</i>	Specifies an unsigned 32-bit integer divisor.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

### Error Conditions

None.

### See Also

*div2x64m*, *div\_cl64m*, *div\_2xcl64m*, *mod64m*, *mod2x64m*.

### Notes

None.

## 3.1.7. div2x64m

### Purpose

Divide an unsigned 64-bit integer by an unsigned 64-bit integer.

### Synopsis

```

#include "u_signed64.h"

u_signed64 div2x64m(
    u_signed64  l11, /* IN */
    u_signed64  l12); /* IN */

```

### Description

The *div2x64m* macro is called to divide an unsigned 64-bit value by an unsigned 64-bit value.

### Parameters

<i>l11</i>	Specifies an unsigned 64-bit integer numerator.
<i>l12</i>	Specifies an unsigned 64-bit integer divisor.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

### Error Conditions

None.

#### See Also

**div64m, div\_cl64m, mod64m, mod2x64m.**

#### Notes

None.

### 3.1.8. div\_cl64m

#### Purpose

Divide an unsigned 64-bit integer by an unsigned 32-bit integer and return the integer ceiling of the result.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 div_cl64m(
    u_signed64  ll1, /* IN */
    unsigned32  ll); /* IN */
```

#### Description

The *div\_cl64m* macro is called to return the integer ceiling resulting from the division of an unsigned 64-bit value by an unsigned 32-bit value.

#### Parameters

<i>ll1</i>	Specifies an unsigned 64-bit integer numerator.
<i>ll</i>	Specifies an unsigned 32-bit integer divisor

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**div64m, div2x64m, div\_2xcl64m, mod64m.**

#### Notes

None.

### 3.1.9. div\_2xcl64m

#### Purpose

Divide an unsigned 64-bit integer by an unsigned 64-bit integer and return the integer ceiling of the result.

#### Synopsis

```
#include "u_signed64.h"
```

```

u_signed64 div_2xcl64m(
    u_signed64  ll1, /* IN */
    u_signed64  ll2); /* IN */

```

### Description

The *div\_2xcl64m* macro is called to return the integer ceiling resulting from the division of an unsigned 64-bit value by an unsigned 64-bit value.

### Parameters

*ll1* Specifies an unsigned 64-bit integer numerator.  
*ll2* Specifies an unsigned 64-bit integer divisor.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

### Error Conditions

None.

### See Also

*div64m*, *div2x64m*, *div\_cl64m*, *mod64m*, *mod2x4m*.

### Notes

None.

## 3.1.10. eqz64m

### Purpose

Determine if an unsigned 64-bit integer is zero.

### Synopsis

```

#include "u_signed64.h"

int eqz64m (
    u_signed64  ll1); /* IN */

```

### Description

The *eqz64m* macro is called to check if an unsigned 64-bit integer equals zero.

### Parameters

*ll1* Specifies the unsigned 64-bit integer.

### Return Values

1 is returned if *ll1* equals zero. Otherwise 0 is returned.

### Error Conditions

None.

#### See Also

**eq64m**, **ge64m**, **gt64m**, **le64m**, **lt64m**, **neqz64m**, **neq64m**.

#### Notes

None.

### 3.1.11. eq64m

#### Purpose

Compare two unsigned 64-bit integers for equality.

#### Synopsis

```
#include "u_signed64.h"

int eq64m (
    u_signed64  ll1, /* IN */
    u_signed64  ll2); /* IN */
```

#### Description

The *eq64m* macro is called to check if two unsigned 64-bit integer values are equal.

#### Parameters

<i>ll1</i>	Specifies the first unsigned 64-bit integer.
<i>ll2</i>	Specifies the second unsigned 64-bit integer.

#### Return Values

1 is returned if *ll1* = *ll2*. Otherwise 0 is returned.

#### Error Conditions

None.

#### See Also

**eqz64m**, **ge64m**, **gt64m**, **le64m**, **lt64m**, **neqz64m**, **neq64m**.

#### Notes

None.

### 3.1.12. ge64m

#### Purpose

Perform greater than or equal check between two unsigned 64-bit integers.

#### Synopsis

```
#include "u_signed64.h"

int ge64m(
    u_signed64  ll1, /* IN */
```



```
u_signed64 ll2); /* IN */
```

### Description

The **ge64m** macro is called to determine if the first unsigned 64-bit integer value is greater than or equal to the second unsigned 64-bit integer.

### Parameters

<i>ll1</i>	Specifies the first unsigned 64-bit integer.
<i>ll2</i>	Specifies the second unsigned 64-bit integer.

### Return Values

1 is returned if *ll1* >=*ll2*. Otherwise 0 is returned.

### Error Conditions

None.

### See Also

**eqz64m**, **eq64m**, **gt64m**, **le64m**, **lt64m**, **neqz64m**, **neq64m**.

### Notes

None.

## 3.1.13. gt64m

### Purpose

Perform greater than check between two unsigned 64-bit integers.

### Synopsis

```
#include "u_signed64.h"

int gt64m(
    u_signed64 ll1, /* IN */
    u_signed64 ll2); /* IN */
```

### Description

The **gt64m** macro is called to determine if the first unsigned 64-bit integer value is greater than the second unsigned 64-bit integer.

### Parameters

<i>ll1</i>	Specifies the first unsigned 64-bit integer.
<i>ll2</i>	Specifies the second unsigned 64-bit integer.

### Return Values

1 is returned if *ll1* > *ll2*. Otherwise 0 is returned.

### Error Conditions

None.

#### See Also

**eqz64m**, **eq64m**, **ge64m**, **le64m**, **lt64m**, **neq64m**.

#### Notes

None.

### 3.1.14. high32m

#### Purpose

Extract the high order 32-bits from an unsigned 64-bit integer.

#### Synopsis

```
#include "u_signed64.h"

unsigned32 high32m(
    u_signed64 lll); /* IN */
```

#### Description

The *high32m* macro is called to extract an unsigned 32-bit integer from the high order 32-bits of an unsigned 64-bit integer.

#### Parameters

*lll*                                Specifies the unsigned 64-bit integer.

#### Return Values

Upon completion, an unsigned 32-bit integer is returned.

#### Error Conditions

None.

#### See Also

**low32m**.

#### Notes

None.

### 3.1.15. le64m

#### Purpose

Perform less than or equal check between two unsigned 64-bit integers.

#### Synopsis

```
#include "u_signed64.h"

int le64m(
```

```

    u_signed64  ll1,  /* IN */
    u_signed64  ll2); /* IN */

```

### Description

The **le64m** macro is called to determine if the first unsigned 64-bit integer value is less than or equal to the second unsigned 64-bit integer.

### Parameters

<i>ll1</i>	Specifies the first unsigned 64-bit integer.
<i>ll2</i>	Specifies the second unsigned 64-bit integer.

### Return Values

1 is returned if *ll1* <= *ll2*. Otherwise 0 is returned.

### Error Conditions

None.

### See Also

**eqz64m**, **eq64m**, **ge64m**, **gt64m**, **lt64m**, **neq64m**.

### Notes

None.

## 3.1.16. low32m

### Purpose

Extract the low order 32-bits from an unsigned 64-bit integer.

### Synopsis

```

#include "u_signed64.h"

unsigned32 low32m(
    u_signed64  ll1); /* IN */

```

### Description

The **low32m** macro is called to extract an unsigned 32-bit integer from the low order 32-bits of an unsigned 64-bit integer.

### Parameters

<i>ll1</i>	Specifies the unsigned 64-bit integer.
------------	--

### Return Values

Upon completion, a 32-bit unsigned integer is returned.

### Error Conditions

None.

#### See Also

**high32m.**

#### Notes

None.

### 3.1.17. lt64m

#### Purpose

Perform less than check between two unsigned 64-bit integers.

#### Synopsis

```
#include "u_signed64.h"

int lt64m(
    u_signed64  ll1, /* IN */
    u_signed64  ll2); /* IN */
```

#### Description

The **lt64m** macro is called to determine if the first unsigned 64-bit integer value is less than the second unsigned 64-bit integer.

#### Parameters

<i>ll1</i>	Specifies the first unsigned 64-bit integer.
<i>ll2</i>	Specifies the second unsigned 64-bit integer.

#### Return Values

1 is returned if  $ll1 < ll2$ . Otherwise 0 is returned.

#### Error Conditions

None.

#### See Also

**eqz64m, eq64m, ge64m, gt64m, le64m, neq64m.**

#### Notes

None.

### 3.1.18. mod64m

#### Purpose

Determine the remainder on division of an unsigned 64-bit integer by an unsigned 32-bit integer.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 mod64m(
```

```

u_signed64 111, /* IN */
unsigned32 11); /* IN */

```

### Description

The **mod64m** macro is called to perform a modulus of an unsigned 64-bit integer value by an unsigned 32-bit integer value.

### Parameters

<i>111</i>	Specifies the unsigned 64-bit integer to divide.
<i>11</i>	Specifies the unsigned 32-bit integer divisor.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

### Error Conditions

None.

### See Also

**div64m**, **div2x64m**, **div\_cl64m**, **div\_2xcl64m**, **mod2x64m**.

### Notes

None.

## 3.1.19. mod2x64m

### Purpose

Determine the remainder on division of an unsigned 64-bit integer by an unsigned 64-bit integer.

### Synopsis

```

#include "u_signed64.h"

u_signed64 mod2x64m(
    u_signed64 111, /* IN */
    u_signed64 112); /* IN */

```

### Description

The **mod2x64m** macro is called to perform a modulus of an unsigned 64-bit integer value by an unsigned 64-bit integer value.

### Parameters

<i>111</i>	Specifies the unsigned 64-bit integer to modulus.
<i>112</i>	Specifies the unsigned 64-bit integer modulus value.

### Return Values

Upon completion, an unsigned 64-bit integer is returned.

### Error Conditions

None.

#### See Also

**div64m, div2x64, div\_cl64m, div\_2xcl64m, mod64m.**

#### Notes

None.

### 3.1.20. mul64m

#### Purpose

Multiply an unsigned 64-bit integer by an unsigned 32-bit integer.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 mul64m(
    u_signed64 111, /* IN */
    unsigned32 11); /* IN */
```

#### Description

The *mul64m* macro is called to multiply an unsigned 64-bit integer value by an unsigned 32-bit integer value.

#### Parameters

<i>111</i>	Specifies the unsigned 64-bit integer multiplier.
<i>11</i>	Specifies the unsigned 32-bit integer multiplier.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

None.

#### Notes

None.

### 3.1.21. neqz64m

#### Purpose

Determine if an unsigned 64-bit integer is nonzero.

#### Synopsis

```
#include "u_signed64.h"
```

```
int neqz64m(
    u_signed64 111); /* IN */
```

#### Description

The *neqz64m* macro is called to check if an unsigned 64-bit integer is nonzero.

#### Parameters

*111* Specifies the unsigned 64-bit integer.

#### Return Values

1 is returned if *111* != 0. Otherwise 0 is returned.

#### Error Conditions

None.

#### See Also

*eqz64m*, *eq64m*, *ge64m*, *gt64m*, *le64m*, *lt64m*, *neq64m*.

#### Notes

None.

### 3.1.22. not64m

#### Purpose

Perform a bitwise NOT of an unsigned 64-bit integer.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 not64m(
    u_signed64 111); /* IN */
```

#### Description

The *not64m* macro is called to perform a bitwise NOT of an unsigned 64-bit integer value.

#### Parameters

*111* Specifies an unsigned 64-bit integer.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

*and64m*, *or64m*.

## Notes

None.

### 3.1.23. or64m

#### Purpose

Find the bitwise OR of two unsigned 64-bit values.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 or64m (
    u_signed64  l11, /* IN */
    u_signed64  l12); /* IN */
```

#### Description

The *or64m* macro is called to perform an OR operation of two unsigned 64-bit integer values.

#### Parameters

<i>l11</i>	Specifies the first unsigned 64-bit integer.
<i>l12</i>	Specifies the second unsigned 64-bit integer.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**and64m, not64m.**

## Notes

None.

### 3.1.24. shl64m

#### Purpose

Shift an unsigned 64-bit integer left by an unsigned 32-bit unsigned integer amount.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 shl64m(
    u_signed64  l11, /* IN */
    unsigned32  l1); /* IN */
```

#### Description



The *shl64m* macro is called to shift an unsigned 64-bit integer left by an unsigned 32-bit integer count.

#### Parameters

<i>lll</i>	Specifies an unsigned 64-bit integer to be shifted.
<i>ll</i>	Specifies an unsigned 32-bit integer shift count.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**shr64m.**

#### Notes

None.

### 3.1.25. shr64m

#### Purpose

Shift an unsigned 64-bit integer right by an unsigned 32-bit integer amount.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 shr64m(
    u_signed64  lll, /* IN */
    unsigned32  ll); /* IN */
```

#### Description

The *shr64m* macro is called to shift an unsigned 64-bit integer right by an unsigned 32-bit integer count.

#### Parameters

<i>lll</i>	Specifies an unsigned 64-bit integer to be shifted.
<i>ll</i>	Specifies an unsigned 32-bit integer shift count.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**shl64m.**

#### Notes

None.

### 3.1.26. sub64m

#### Purpose

Subtract two unsigned 64-bit integers.

#### Synopsis

```
#include "u_signed64.h"

u_signed64 sub64m(
    u_signed64 111, /* IN */
    u_signed64 112); /* IN */
```

#### Description

The *sub64m* macro is called to subtract two unsigned 64-bit integer values.

#### Parameters

<i>111</i>	Specifies an unsigned 64-bit integer.
<i>112</i>	Specifies an unsigned 64-bit integer to subtract from the first operand.

#### Return Values

Upon completion, an unsigned 64-bit integer is returned.

#### Error Conditions

None.

#### See Also

**sub64\_3m.**

#### Notes

None.

### 3.1.27. sub64\_3m

#### Purpose

Subtract two unsigned 64-bit integers and store the result in a separate integer field.

#### Synopsis

```
#include "u_signed64.h"

void sub64m_3m(
    u_signed64 111, /* OUT */
    u_signed64 112, /* IN */
    u_signed64 113); /* IN */
```

#### Description

The ***sub64\_3m*** macro is called to subtract two unsigned 64-bit integers and store the result into a third unsigned 64-bit integer.

#### Parameters

<i>ll1</i>	Specifies the result of the subtraction of the 2 unsigned 64-bit operands.
<i>ll2</i>	Specifies the first unsigned 64-bit integer subtraction operand.
<i>ll3</i>	Specifies an unsigned 64-bit integer to subtract from the first operand.

#### Return Values

None. The result of the subtraction is stored in the first parameter.

#### Error Conditions

None.

#### See Also

**sub64m.**

#### Notes

None.

## 3.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

### 3.2.1. u\_signed64

#### Description

u\_signed64 is the type for an unsigned 64-bit integer.

#### Format

For big-endian platforms, the u\_signed64 type is defined in the u\_signed64.h file as follows:

```
typedef struct{
    unsigned long h;
    unsigned long l;
} u_signed64;
```

For little-endian platforms, the u\_signed64 type is defined in the u\_signed64.h file as follows:

```
typedef struct{
    unsigned long l;
    unsigned long h;
} u_signed64;
```

### 3.2.2. unsigned32

#### Description

unsigned32 is the type for an unsigned 32-bit integer.

#### Format

The unsigned32 type is defined in the u\_signed64.h file as follows:

```
typedef unsigned long unsigned32;
```

## Chapter 4. Site Interfaces

This chapter describes all Site Interfaces which are provided for use by two shared libraries (Account Validation site policy and Gatekeeping site policy) which will each be dynamically loaded with the Gatekeeper Server. The Site interface specification includes the following information:

- Name
- Purpose
- Synopsis
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Clients
- Notes
- Example Usage

### 4.1. Gatekeeping

This section describes the Gatekeeping Site Interfaces. Function definitions for these Site Interfaces are provided with the HPSS release, thus the Gatekeeper Server installed by default does NO gatekeeping. Sites will need to enhance these interfaces to implement local policy rules. The site interface code exists in `/opt/hpss/src/sitelib/gk`. Each Site Interface will be called by a Gatekeeping Service API. Please refer to the "Example Usage" section of each Site Interface for example "pseudo code" implementation details of a site defined policy.

Please note that the code listed here is merely to give an idea of example uses. It has not been tested in an actual system.

#### 4.1.1. `gk_site_Close`

##### Purpose

This is an internal, site defined-and-implemented function that is called by the Gatekeeper whenever a file is closed. This function is not an RPC. It is a call to a procedure in a shared library.

##### Synopsis

```
signed32
gk_site_Close (
    hpss_uid_t      ControlNo);    /* IN */
```

##### Description

This function is written by the customer site. It will be called by `gk_Close` while processing a close of an HPSS file.

## Parameters

<i>ControlNo</i>	Unique identifier for the opened file which is now being closed. This number was generated by the Gatekeeper and passed into <b>gk_site_Open</b> or <b>gk_site_OpenStats</b> .
------------------	--

## Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

## Error Conditions

The **gk\_site\_Close** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOENT	Could not find an entry matching the ControlNo field in <i>EntryInfo</i> .
HPSS_ENOTREADY	Initialization has not completed.

## See Also

**gk\_Close**, **gk\_Open**, **gk\_site\_Open**, and **gk\_site\_OpenStats**.

## Clients

Gatekeeper.

## Notes

None.

## Example Usage

Log entrance of this routine to the Site Policy Logfile.

Lock the Request Cache.

Lookup the ControlNo in the Request Cache.

If not found, then

    It must have already been removed.

NOTE: There is a situation in which this will happen.

    It could be that the GK was bounced while the Core Server was  
    handling the termination. So, don't do anything rash.

Unlock the Request Cache.

Log an Error to the Site Policy Logfile.

return.

If the Request was NOT an AuthorizedCaller request, then

    If the State of this request is GOOD, then

        Decrement the Host Count of the good opens pending.

Decrement the User Count of the good opens pending.

Decrement the global number of good opens pending.

Else if the State of this request is RETRY, then

Decrement the global number of retry opens pending.

If the Request was in a retry state due to the host policy

    denying the open, then

        Decrement the Host's Count of the retry opens pending.

If the Request was in a retry state due to the user policy

    denying the open, then

        Decrement the User's Count of the retry opens pending.

Delete the Request from the Request Cache.

Unlock the Request Cache.

Log exit of this routine to the Site Policy Logfile.

### 4.1.2. **gk\_site\_Create**

#### **Purpose**

This is an internal, site defined-and-implemented function that implements site policies which decide if a caller is authorized to create the file now, later, or not at all. This function is not an RPC. It is a call to a procedure in a shared library.

#### **Synopsis**

```
signed32
gk_site_Create (
    gk_EntryInfo_t      EntryInfo,          /* IN */
    unsigned32          *WaitTimeP)         /* OUT */
```

#### **Description**

This function is written by the customer site. It is called by **gk\_Create** whenever an HPSS file create occurs.

#### **Parameters**

<i>EntryInfo</i>	Information about the file being created.
<i>WaitTimeP</i>	A pointer to the number of seconds to wait before retrying a request.

#### **Return Values**

Upon successful completion, a value of zero (0) is returned which tells the Gatekeeper to continue with the create request. After the Gatekeeper has returned success to the caller, it expects **gk\_site\_CreateComplete** to be called when the file has been created or if an error occurs.

All non-zero return values will be treated as errors, thus the client will do the appropriate error handling and terminate the create request. Non-zero error values are described in the Error Conditions.

## Error Conditions

The **gk\_site\_Create** routine is unsuccessful if any of the following are true:

HPSS_EACCES	Permission is denied.
HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOTREADY	Initialization has not completed.
HPSS_EPERM	The operation is not permitted.
HPSS_ERETRY	The operation should be retried after the delay, in seconds, returned by the <i>WaitTimeP</i> parameter.
HPSS_ETHRESHOLD_DENY	This error code is returned to deny access due to exceeding a threshold. The Core Server will return this error to the Client API. The Client API will map this error into EBUSY before returning the error code to the application.
HPSS_EUSER_DENY	This error code is returned to deny access to a particular user. The Core Server will return this error to the Client API. The Client API will map this error into EACCESS before returning the error code to the application.

## See Also

**gk\_Create**, **gk\_CreateComplete**, **gk\_site\_CreateComplete** and **gk\_site\_CreateStats**.

## Clients

Gatekeeper.

## Notes

Routine **gk\_site\_CreateStats** is called instead of **gk\_site\_Create** whenever an HPSS file create occurs from an authorized caller when authorized caller requests are monitored.

If the operation should be retried, HPSS\_RETRY is returned and the number of seconds the caller should wait before retrying is returned in the *WaitTimeP* parameter. If the site does not specify the *WaitTimeP* parameter (returns 0), the default value from the Gatekeeper's specific configuration field DefaultWaitTime is used instead.

## Example Usage

Log entrance of this routine to the Site Policy Logfile using the

    RequestId passed in the EntryInfo.

Initialize the WaitTimeP return parameter to 0.

If we're not monitoring for creates, then

    Log an ALARM Error to the Site Policy Logfile.

    Return, but don't return an error status -- just let the create continue.

If there is a no Site Policy file

    Log an Error to the Site Policy Logfile.



Return good status.

Lock the Request Cache.

Find the Request in the Request Cache.

If not found, then

    Create a new Request Entry and add it to the Request Cache.

Else

    The Request is being retried.

If there exists a "Maximum Number of Creates Per Host Policy", then

    Note: Translating socket address into hostnames is expensive, so

        maintain a cache to translate HostAddrs to HostNames.

    Lock the HostAddrToHostName Cache.

    Lookup the EntryInfo.HostAddr in the HostAddrToHostName Cache.

    If it doesn't exist, then

        hpss\_GetHostByAddr

        If Error returned by multithreaded safe get host by addr lookup, then

            Log an error to the Site Policy Logfile.

            Delete the Request from the Request Cache.

            Unlock all locks.

            Return HPSS\_EFAULT.

        Create, initialize and add a new HostAddrToHostName cache entry.

        Save away the HostName.

        Unlock the HostAddrToHostName Cache.

    Lookup the HostName in the Host Entry Cache.

    Note: The Host Entry Cache is sharing the Request Cache Lock.

    If not found, then

        Create a new Host Entry and add it to the Host Cache.

    If the Host Entry's good creates pending  $\geq$  Maximum Creates Per Host,

    then

        Mark this request as a RETRY due to host policy denial.

        Set Error to HPSS\_ERETRY.

If the Request was not originally a RETRY, then  
Increment the retry creates pending in the Host Entry.

If there exists a "Maximum Number of Creates Per User Policy" AND Error  
wasn't set in the Host Policy code above, then

Lookup the UserId/RealmId pair in the User Entry Cache.

Note: The UserId and RealmId together will uniquely identify a  
particular user in a federated name space (cross realm)  
environment.

Note: The User Entry Cache is sharing the Request Cache Lock.

If not found, then

Create a new User Entry and add it to the User Cache.

If the User Entry's good creates pending  $\geq$  Maximum Creates Per User,  
then

Mark this request as a RETRY due to user policy denial.

Set Error to HPSS\_ERETRY.

If the Request was not originally a RETRY, then

Increment the retry creates pending in the User Entry.

If we want to return HPSS\_ERETRY, then

If this originally wasn't a retry request (i.e. first delay of request),  
then

Increment the global retry creates pending count.

Increment the RetryCount associated with the Request Entry.

Mark the Request Entry's state as RETRY.

Compute the WaitTime corresponding to the number of seconds to wait  
before the Client API will retry this request.

Else if no Error, then

Increment the global good creates pending count.

Mark the Request Entry's state as GOOD.

If this request is being retried again, then

- Decrement the global retry creates pending count.

If Host policy exists, then

- Decrement the Host Entry's good creates pending count.

If this request was previously retried due to host policy denial, then

- Decrement the Host Entry's retry creates pending count.

If User policy exists, then

- Decrement the User Entry's good creates pending count.

If this request was previously retried due to user policy denial, then

- Decrement the Host User's retry creates pending count.

Unlock the Request Cache.

Log exit of this routine to the Site Policy Logfile.

Return the status.

### 4.1.3. **gk\_site\_CreateComplete**

#### **Purpose**

This is an internal, site defined-and-implemented function that is called by the Gatekeeper when a file create has completed. This function is not an RPC. It is a call to a procedure in a shared library.

#### **Synopsis**

```
signed32
gk_site_CreateComplete (
    hpss_uuid_t      ControlNo);      /* IN */
```

#### **Description**

This function is written by the customer site. It will be called by **gk\_CreateComplete** while processing a file create completion of an HPSS file.

#### **Parameters**

<i>ControlNo</i>	Unique identifier for the created file which is now being completed. This number was generated by the Gatekeeper and passed into <b>gk_site_Create</b> or <b>gk_site_CreateStats</b> .
------------------	--

#### **Return Values**

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

#### **Error Conditions**

The **gk\_site\_CreateComplete** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOENT	Could not find an entry matching the ControlNo field in <i>EntryInfo</i> .
HPSS_ENOTREADY	Initialization has not completed.

#### See Also

**gk\_Create**, **gk\_CreateComplete**, **gk\_site\_Create** and **gk\_site\_CreateStats**.

#### Clients

Gatekeeper.

#### Notes

None.

#### Example Usage

Please see **gk\_site\_Close**.

### 4.1.4. gk\_site\_CreateStats

#### Purpose

This is an internal, site defined-and-implemented function that is called asynchronously by the Gatekeeper when an authorized caller is creating a file. It is only called when authorized caller requests and create requests are being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

#### Synopsis

```
void
gk_site_CreateStats (
    gk_EntryInfo_t    EntryInfo);    /* IN */
```

#### Description

This function is written by the customer site. It is called asynchronously by **gk\_Create** whenever an HPSS file create occurs from an authorized caller when both create requests and authorized caller requests are monitored.

#### Parameters

<i>EntryInfo</i>	Information about the file being created.
------------------	---

#### Return Values

None.

#### Error Conditions

None.

#### See Also

**gk\_Create**, **gk\_CreateComplete**, **gk\_site\_Create** and **gk\_site\_CreateComplete**.

#### Clients

Gatekeeper.

## Notes

Routine **gk\_site\_CreateStats** is called instead of **gk\_site\_Create** whenever an HPSS file create occurs from an authorized caller when authorized caller requests are monitored.

The Gatekeeper will queue all asynchronous calls to the site interfaces so that they are processed in the order they are received. For example, if a site is monitoring authorized caller, create and open requests and an authorized caller create request is issued before an authorized caller open request, then the Gatekeeper will queue these requests so that **gk\_site\_CreateStats** is called before **gk\_site\_OpenStats**.

## Example Usage

Log entrance of this routine to the Site Policy Logfile using the

    RequestId passed in the EntryInfo.

If we're not monitoring for creates, then

    Log an ALARM Error to the Site Policy Logfile.

    Return.

If there is a no Site Policy file

    Log an Error to the Site Policy Logfile.

    Return.

Lock the Request Cache.

Find the Request in the Request Cache.

If not found, then

    Create a new Request Entry and add it to the Request Cache.

If there exists a "Maximum Number of Creates Per Host Policy", then

    Note: Translating socket address into hostnames is expensive, so

        maintain a cache to translate HostAddrs to HostNames.

    Lock the HostAddrToHostName Cache.

    Lookup the EntryInfo.HostAddr in the HostAddrToHostName Cache.

    If it doesn't exist, then

        hpss\_GetHostByAddr

    If Error returned by multithreaded safe get host by addr lookup, then

        Log an error to the Site Policy Logfile.

Delete the Request from the Request Cache.

Unlock all locks.

Return.

Create, initialize and add a new HostAddrToHostName cache entry.

Save away the HostName.

Unlock the HostAddrToHostName Cache.

Lookup the HostName in the Host Entry Cache.

Note: Assume the Host Entry Cache is sharing the Request Cache Lock.

If not found, then

Create a new Host Entry and add it to the Host Cache.

If there exists a "Maximum Number of Creates Per User Policy", then

Lookup the UserId/RealmId pair in the User Entry Cache.

Note: The UserId and RealmId together will uniquely identify a particular user in a federated name space (cross realm) environment.

Note: The User Entry Cache is sharing the Request Cache Lock.

If not found, then

Create a new User Entry and add it to the User Cache.

Mark the Request Entry's state as GOOD.

Unlock the Request Cache.

Log exit of this routine to the Site Policy Logfile.

Return.

### 4.1.5. gk\_site\_GetMonitorTypes

#### Purpose

This is an internal, site defined-and-implemented function that is called by the Gatekeeper get the types of items being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

#### Synopsis

**signed32**

```
gk_site_GetMonitorTypes (
    u_signed64      MonitorTypeBitsP); /* OUT */
```

## Description

This interface will lookup the local site policies to determine which request types are being monitored. It will then return a bitvector of items being monitored. The Core Server will call **gk\_GetMonitorTypes** when (re)connecting to the Gatekeeper to learn which request types are being monitored so that it can call the Gatekeeper appropriately. The Gatekeeper will call **gk\_site\_GetMonitorTypes**. Currently the following request types can be monitored: authorized caller, create, open, and stage.

It is important that the Site Interfaces return a status in a timely fashion. Create, open, and stage requests from XFS, NFS, and MPS (authorized callers) are timing sensitive, thus the Site Interfaces won't be permitted to delay or stop these requests, however the Site Interfaces may choose to be involved in keeping statistics on these requests by monitoring requests from authorized callers.

## Parameters

<i>MonitorTypeBitsP</i>	A bit vector (0 origin array of bits) in which the appropriate bit is set (on) for each request type that is currently being monitored. Request types index are:
	GK_MONITOR_AUTHORIZED_CALLER      0
	GK_MONITOR_CREATE                    1
	GK_MONITOR_OPEN                      2
	GK_MONITOR_STAGE                    3

## Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

## Error Conditions

The **gk\_site\_GetMonitorTypes** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
-------------	--

## See Also

**gk\_GetMonitorTypes.**

## Clients

Gatekeeper

## Notes

The following table describes which Site Interface will be called for each request type when authorized caller requests are monitored.

Request Type	Authorized Caller	All Other Users
Create	gk_site_CreateStats	gk_site_Create
Open	gk_site_OpenStats	gk_site_Open
Stage	gk_site_StageStats	gk_site_Stage

### Example Usage

Log entrance of this routine to the Site Policy Logfile.

Set the bits corresponding to what we're monitoring in the MonitorTypeBitsP.

(e.g. \*MonitorTypeBitsP = orbit64m(\*MonitorTypeBitsP, GK\_MONITOR\_OPEN))

Log exit of this routine to the Site Policy Logfile.

Return status.

## 4.1.6. gk\_site\_Init

### Purpose

This is an internal, site defined and implemented function that is called when the Gatekeeper is (re) initialized. This function is used to do whatever initialization is needed by the site module. This function is not an RPC. It is a call to a procedure in a shared library.

### Synopsis

```
signed32
gk_site_Init (
    char                *SitePolicyPathNameP ); /* IN */
```

### Description

This function is written by the customer site. It will be called by the Gatekeeper's internal initialization routine whenever the Gatekeeper initializes or reinitializes.

### Parameters

<i>SitePolicyPathNameP</i>	A pointer to the path name of the file where the site policy is stored. This path name is defined to be HPSS_MAX_PATH_NAME bytes long which is 1024 bytes.
----------------------------	--

### Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

### Error Conditions

The **gk\_site\_Init** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
-------------	--

### See Also

None.

### Clients

Gatekeeper.

### Notes



If an error value is returned, the Gatekeeper Server will not initialize.

### Example Usage

Initialize mutex locks, global variables, etc.

If the passed in SitePolicyPathNameP is non null and not nil, then

Read the site policy file looking for things like:

- The types of requests we're monitoring (authorized caller, creates, opens, stages)
- Pathname to a log file.
- Maximum size of the log file.
- Debug settings.
- Pathname to a dump file.
- Maximum number of creates per host.
- Maximum number of creates per user.
- Maximum number of opens per host.
- Maximum number of opens per user.
- Maximum number of stages per host. (Note: Needs to be greater than the maximum number of opens per host.)
- Maximum number of stages per user. (Note: Needs to be greater than the maximum number of opens per user.)

Open and initialize the logfile (if one is configured).

Log exit of this routine to the Site Policy Logfile.

Return status.

## 4.1.7. gk\_site\_Open

### Purpose

This is an internal, site defined-and-implemented function that implements site policies which decide if a caller is authorized to open the file now, later, or not at all. This function is not an RPC. It is a call to a procedure in a shared library.

### Synopsis

```
signed32
gk_site_Open (
    gk_EntryInfo_t    EntryInfo,    /* IN */
    unsigned32        *WaitTimeP); /* OUT */
```

### Description

This function is written by the customer site. It is called by **gk\_Open** whenever an HPSS file open occurs.

### Parameters

<i>EntryInfo</i>	Information about the file being opened.
<i>WaitTimeP</i>	A pointer to the number of seconds to wait before retrying a request.

### Return Values

Upon successful completion, a value of zero (0) is returned which tells the Gatekeeper to continue with the open request. After the Gatekeeper has returned success to the caller, it expects **gk\_site\_Close** to be called when the file has been closed or if an error occurred

All non-zero return values will be treated as errors, thus the client will do the appropriate error handling and terminate the open request. Non-zero error values are described in the Error Conditions.

### Error Conditions

The **gk\_site\_Open** routine is unsuccessful if any of the following are true:

HPSS_EACCES	Permission is denied.
HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOTREADY	Initialization has not completed.
HPSS_EPERM	The operation is not permitted.
HPSS_ERETRY	The operation should be retried after the delay, in seconds, returned by the <i>WaitTimeP</i> parameter.
HPSS_ETHRESHOLD_DENY	This error code is returned to deny access due to exceeding athreshold. The Core Server will return this error to the Client API. The Client API will map this error into EBUSY before returning the error code to the application.
HPSS_EUSER_DENY	This error code is returned to deny access to a particular user. The Core Server will return this error to the Client API. The Client API will map this error into EACCESS before returning the error code to the application.

### See Also

**gk\_Close**, **gk\_Open**, **gk\_site\_Close**, and **gk\_site\_OpenStats**.

### Clients

Gatekeeper.

### Notes

Routine **gk\_site\_OpenStats** is called instead of **gk\_site\_Open** whenever an HPSS file open occurs from an authorized caller when authorized caller requests are monitored.

If the operation should be retried, HPSS\_RETRY is returned and the number of seconds the caller should wait before retrying is returned in the *WaitTimeP* parameter. If the site does not specify the *WaitTimeP* parameter (returns 0), the default value from the Gatekeeper's specific configuration field DefaultWaitTime is used instead.

## Example Usage

Please see section **gk\_site\_Create**.

## 4.1.8. gk\_site\_OpenStats

### Purpose

This is an internal, site defined-and-implemented function that is called asynchronously by the Gatekeeper when an authorized caller is opening a file. It is only called when authorized caller requests and open requests are being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

### Synopsis

```
void
gk_site_OpenStats (
    gk_EntryInfo_t    EntryInfo);    /* IN */
```

### Description

This function is written by the customer site. It is called by **gk\_Open** whenever an HPSS file open occurs from an authorized caller when both open requests and authorized caller requests are monitored.

### Parameters

<i>EntryInfo</i>	Information about the file being opened.
------------------	--

### Return Values

None.

### Error Conditions

None.

### See Also

**gk\_Close**, **gk\_Open**, **gk\_site\_Close**, and **gk\_site\_Open**.

### Clients

Gatekeeper.

### Notes

Routine **gk\_site\_OpenStats** is called instead of **gk\_site\_Open** whenever an HPSS file open occurs from an authorized caller when authorized caller requests are monitored.

The Gatekeeper will queue all asynchronous calls to the site interfaces so that they are processed in the order they are received. For example, if a site is monitoring authorized caller, create and open requests and an authorized caller create request is issued before an authorized caller open request, then the Gatekeeper will queue these requests so that **gk\_site\_CreateStats** is called before **gk\_site\_OpenStats**.

## Example Usage

Please see section **gk\_site\_CreateStats**.

## 4.1.9. gk\_site\_PassThru

### Purpose

This is an internal, site defined-and-implemented function that is called by the Gatekeeper to pass information to the site interface. This function is not an RPC. It is a call to a procedure in a shared library.

### Synopsis

```
signed32
gk_site_PassThru (
    u_signed64      Param64,          /* IN */
    unsigned char   *ParamString);   /* IN */
```

### Description

This function is written by the customer site. It will be used to pass information into the site interface. The site interface can define what these parameters mean.

This function will be called by **gk\_PassThru**.

### Parameters

<i>Param64</i>	A 64 bit parameter.
<i>ParamString</i>	NULL-terminated string parameter.

### Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

### Error Conditions

The **gk\_site\_PassThru** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOTREADY	Initialization has not completed.

### See Also

**gk\_PassThru**.

### Clients

Gatekeeper.

### Notes

The site may want to define bits in *Param64* to correspond to special site actions. The *ParamString* may be used in conjunction with a bit(s) in *Param64* or for some other use. For example, if bit position 0 is ON in *Param64* then dump the current state to some file name passed in *ParamString*. The site will need to write its own client to use this interface (i.e. Core Server and SSM will not use this interface).

### Example Usage

Log entrance of this routine to the Site Policy Logfile.

If *Param64* is zero, then

Log exit of this routine to the Site Policy Logfile.

Return status.

Verify bits passed into Param64 are valid options.

If an unsupported bit is set in Param64, then

Log an exit error to the Site Policy Logfile.

Return status.

If turned on the DumpState bit, then

If ParamString is null or nil, then

Dump the state of each cache and all the other good information into  
the dump file.

Else

Dump the state of each cache and all the other good information into  
the pathname stored in ParamString.

Log exit of this routine to the Site Policy Logfile.

Return status.

## 4.1.10. **gk\_site\_ReadSitePolicy**

### **Purpose**

This is an internal, site defined-and-implemented function that is called by the Gatekeeper to inform the site interface that the site policy file needs to be read. This function is not an RPC. It is a call to a procedure in a shared library.

### **Synopsis**

```
signed32  
gk_site_ReadSitePolicy (void);
```

### **Description**

This function is written by the customer site. It will be used to inform the site interface that the site policy file has been modified. This will allow the site interface to re-read the site policy file, if defined, and enforce the new policy.

If the monitoring types are changed, the Gatekeeper and Core Server will need to be restarted in order for the Core Server to learn about it. Therefore, ignore changes in monitor type policy.

This function will be called by **gk\_ReadSitePolicy**.

### **Parameters**

None.

### **Return Values**

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error

Conditions.

## Error Conditions

The **gk\_site\_ReadSitePolicy** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOTREADY	Initialization has not completed.

## See Also

**gk\_ReadSitePolicy**, and **gk\_GetMonitorTypes**.

## Clients

Gatekeeper.

## Notes

If the monitoring types are changed, the Gatekeeper and Core Server will need to be restarted in order for the Core Server to learn about it. Therefore, ignore changes in monitor type policy.

## Example Usage

Log entrance of this routine to the Site Policy Logfile.

If a site policy file exists, then

Read the site policy file looking for things like:

- The types of requests we're monitoring (authorized caller, creates, opens, stages)
- Pathname to a log file.
- Maximum size of the log file.
- Debug settings.
- Pathname to a dump file.
- Maximum number of creates per host.
- Maximum number of creates per user.
- Maximum number of opens per host.
- Maximum number of opens per user.
- Maximum number of stages per host. (Note: Needs to be greater than the maximum number of opens per host.)
- Maximum number of stages per user. (Note: Needs to be greater than the maximum number of opens per user.)

Close the logfile (if one is already opened).

Open the possibly new logfile (if one is configured).

Log exit of this routine to the Site Policy Logfile.

Return status.

## 4.1.11. **gk\_site\_Shutdown**

### **Purpose**

This is an internal, site defined-and-implemented function that is called when the Gatekeeper is shutdown. This function is used to do whatever shutdown and cleanup is needed by the site module. This function is not an RPC. It is a call to a procedure in a shared library.

### **Synopsis**

```
signed32  
gk_site_Shutdown (void);
```

### **Description**

This function is written by the customer site. It will be called by the Gatekeeper's internal shutdown routine whenever the Gatekeeper is issued a shutdown request.

### **Parameters**

None.

### **Return Values**

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

### **Error Conditions**

The **gk\_site\_Shutdown** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
-------------	--

### **See Also**

None.

### **Clients**

Gatekeeper.

### **Notes**

The Gatekeeper Server will not call this routine if it is issued a halt request or is crashing due to some critical problem.

### **Example Usage**

Log entrance of this routine to the Site Policy Logfile.

Flush and close the logfile, if one is already opened.

Return status.

## 4.1.12. gk\_site\_Stage

### Purpose

This is an internal, site defined-and-implemented function that implements site policies which decide if a caller is authorized to stage the file now, later, or not at all. This function not an RPC. It is a call to a procedure in a shared library.

### Synopsis

```
signed32
gk_site_Stage (
    gk_EntryInfo_t      EntryInfo,      /* IN */
    unsigned32          *WaitTimeP);    /* OUT */
```

### Description

This function is written by the customer site. It is called by **gk\_Stage** whenever an HPSS file stage occurs.

### Parameters

<i>EntryInfo</i>	Information about the file being staged.
<i>WaitTimeP</i>	A pointer to the number of seconds to wait before retrying a request.

### Return Values

Upon successful completion, a value of zero (0) is returned which tells the Gatekeeper to continue with the stage request. After the Gatekeeper has returned success to the caller, it expects **gk\_site\_StageComplete** to be called when the file has been staged or if an error occurred.

All non-zero return values will be treated as errors, thus the client will do the appropriate error handling and terminate the stage request. Non-zero error values are described in the Error Conditions.

### Error Conditions

The **gk\_site\_Stage** routine is unsuccessful if any of the following are true:

HPSS_EACCES	Permission is denied.
HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOTREADY	Initialization has not completed.
HPSS_EPERM	The operation is not permitted.
HPSS_ERETRY	The operation should be retried after the delay, in seconds, returned by the <i>WaitTimeP</i> parameter.
HPSS_ETHRESHOLD_DENY	This error code is returned to deny access due to exceeding a threshold. The Core Server will return this error to the Client API. The Client API will map this error into EBUSY before returning the error code to the application.
HPSS_EUSER_DENY	This error code is returned to deny access to a particular user. The Core Server will return this error to the Client API. The Client API will map this error into EACCESS before returning the error code to the application.

### See Also



**gk\_site\_StageComplete**, **gk\_site\_StageStats**, **gk\_Stage**, and **gk\_StageComplete**.

#### Clients

Gatekeeper.

#### Notes

Routine **gk\_site\_StageStats** is called instead of **gk\_site\_Stage** whenever an HPSS file stage occurs from an authorized caller when authorized caller requests are monitored.

If the operation should be retried, HPSS\_RETRY is returned and the number of seconds the caller should wait before retrying is returned in the *WaitTimeP* parameter. If the site does not specify the *WaitTimeP* parameter (returns 0), the default value from the Gatekeeper's specific configuration field DefaultWaitTime is used instead.

#### Example Usage

Please see section **gk\_site\_Create**.

### 4.1.13. gk\_site\_StageComplete

#### Purpose

This is an internal, site defined-and-implemented function that is called by the Gatekeeper when a file stage has completed. This function is not an RPC. It is a call to a procedure in a shared library.

#### Synopsis

```
signed32
gk_site_StageComplete (
    hpss_uuid_t      ControlNo);    /* IN */
```

#### Description

This function is written by the customer site. It will be called by **gk\_StageComplete** while processing a file stage completion of an HPSS file.

#### Parameters

<i>ControlNo</i>	Unique identifier for the staged file which is now being completed. This number was generated by the Gatekeeper and passed into <b>gk_site_Stage</b> or <b>gk_site_StageStats</b> .
------------------	---

#### Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

#### Error Conditions

The **gk\_site\_StageComplete** routine is unsuccessful if any of the following are true:

HPSS_EAGAIN	Resources are temporarily unavailable.
HPSS_ENOENT	Could not find an entry matching the ControlNo field in <i>EntryInfo</i> .
HPSS_ENOTREADY	Initialization has not completed.

#### See Also

**gk\_site\_Stage**, **gk\_site\_StageStats**, **gk\_Stage**, and **gk\_StageComplete**.

#### Clients

Gatekeeper.

#### Notes

None.

#### Example Usage

Please see section **gk\_site\_Close**.

## 4.1.14. gk\_site\_StageStats

#### Purpose

This is an internal, site defined-and-implemented function that is called asynchronously by the Gatekeeper when an authorized caller is staging a file. It is only called when authorized caller requests and stage requests are being monitored. This function is not an RPC. It is a call to a procedure in a shared library.

#### Synopsis

```
void
gk_site_StageStats (
    gk_EntryInfo_t    EntryInfo) ;          /* IN */
```

#### Description

This function is written by the customer site. It is called by **gk\_Stage** whenever an HPSS file stage occurs from an authorized caller when both stage requests and authorized caller requests are monitored.

#### Parameters

*EntryInfo*                      Information about the file being staged.

#### Return Values

None.

#### Error Conditions

None.

#### See Also

**gk\_site\_Stage**, **gk\_site\_StageComplete**, **gk\_Stage**, and **gk\_StageComplete**.

#### Clients

Gatekeeper.

#### Notes

Routine **gk\_site\_StageStats** is called instead of **gk\_site\_Stage** whenever an HPSS file stage occurs from an authorized caller when authorized caller requests are monitored.

The Gatekeeper will queue all asynchronous calls to the site interfaces so that they are processed in the order they are received. For example, if a site is monitoring authorized caller, create and open requests and an authorized caller create request is issued before an authorized caller open request, then the Gatekeeper will queue these requests so that **gk\_site\_CreateStats** is called before **gk\_site\_OpenStats**.

### Example Usage

Please see section **gk\_site\_CreateStats**.

## 4.2. Account Validation Site Interface

This section describes the site customizable parts of the Account Validation APIs. These routines are stored in a shared library to ease customization.

### 4.2.1. av\_site\_AcctIdxToName

#### Purpose

Customizable routine to convert an account index to an account name.

#### Synopsis

```
signed32
av_site_AcctIdxToName(
hpss_connect_handle_t  Binding,           /* IN */
hpss_reqid_t           RequestId,        /* IN */
unsigned32              RealmId,         /* IN */
unsigned32              Uid,             /* IN */
unsigned32              Gid,             /* IN */
unsigned32              Flags,           /* IN */
acct_rec_t             Acct,           /* IN */
char                   *AcctName,       /* OUT */
unsigned32              *OkToCache       /* OUT */
);
```

#### Description

This site customizable routine converts from an account index to an account name. By default, this routine is nothing more than a skeleton that returns HPSS\_EUSEDEFAULT immediately.

#### Parameters

<i>Binding</i>	Binding handle to this server.
<i>RequestId</i>	Request identifier to use when logging messages for this request.
<i>RealmId</i>	Realm identifier of this account.
<i>Uid</i>	User id (UID) of user.
<i>Gid</i>	Group id (GID) of user.
<i>Flags</i>	May be zero or more of the following OR'd together:  ACCT_FLAGS_VALID_INDEX – Verify that the specified user is allowed to use this account index.
<i>Acct</i>	Account Index to translate.

<i>AcctName</i>	Pointer to returned account name.
<i>OkToCache</i>	True if Account Validation Library should cache the result. See Notes for more information.

### Return Values

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

### Error Conditions

HPSS\_EUSEDEFAULT      Tells the caller to use the default built-in behavior for this routine.

Other suggested error codes to use are:

HPSS\_EBUSY              The server is busy. The request should be retried later.

HPSS\_ENOENT            The account in Acct does not exist. AcctName is set to the string form of Acct. If ACCT\_FLAGS\_VALID\_INDEX is set in the Flags parameter, you should return HPSS\_EPERM instead.

HPSS\_EPERM             The specified user is not allowed to use that account index.

HPSS\_ESYSTEM           This routine received an internal error. Usually a site customization problem.

### See Also

**av\_AcctIdxToName**

### Clients

Client API via Gatekeeper

### Notes

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted.

The *OkToCache* argument should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned information can change over time and should not be cached by the client.

### Example Usage

This example is the default behavior of the **av\_AcctIdxToName** routine. See the implementation of that routine, if available, for more details.

```
av_AcctIdxToName() {
    if site style accounting {
        if Acct == ACCT_REC_DEFAULT
            lookup user default acct;
        else lookup account Acct;
        if (not found) {
            if Flags & ACCT_FLAGS_VALID_INDEX
```

```

        error = HPSS_EPERM;
    else error = HPSS_ENOENT;
}

    else if (unix style accounting)
        if (Acct == ACCT_REC_DEFAULT) Acct = Uid;
        if (Acct != Uid && Flags & ACCT_FLAGS_VALID_INDEX)
            error = HPSS_EPERM;
        else {
            convert Realm,Acct into name
            if not found && Flags & ACCT_FLAGS_VALID_INDEX
                error = HPSS_EPERM;
        }
    }
    return error;
}

```

## 4.2.2. av\_site\_AcctNameToldx

### Purpose

Customizable routine to convert an account name to an account index

### Synopsis

```

signed32
av_site_AcctNameToIdx(
hpss_connect_handle_t  Binding,           /* IN */
hpss_reqid_t           RequestId,         /* IN */
unsigned32              RealmId,          /* IN */
unsigned32              Uid,              /* IN */
unsigned32              Gid,              /* IN */
unsigned32              Flags,            /* IN */
char                    *InAcctName,      /* IN */
char                    *OutAcctName,     /* OUT */
acct_rec_t              *Acct,           /* OUT */
unsigned32              *OkToCache        /* OUT */
);

```

### Description

This site customizable routine converts from an account name to an account index. By default, this routine is nothing more than a skeleton that returns HPSS\_EUSEDEFAULT immediately.

### Parameters

<i>Binding</i>	Binding handle to this server.
<i>RequestId</i>	Request Id to use when logging messages for this request.
<i>RealmId</i>	Realm identifier of user.
<i>Uid</i>	User id (UID) of user.
<i>Gid</i>	Group id (GID) of user.
<i>Flags</i>	May be zero or more of the following OR'd together:  ACCT_FLAGS_VALID_INDEX – Verify that the specified user is allowed to use this account index.
<i>InAcctName</i>	Optional pointer to account name to translate. If this argument is NULL or points to an empty string, information about the user's default account index is returned.
<i>OutAcctName</i>	Optional pointer to returned account name. The name of the user's default account is returned if the InAcctName parameter is NULL or points to an empty string. Otherwise, the value of InAcctName is returned. Note that the name of the default account for UNIX Style accounting is the empty string.
<i>Acct</i>	Pointer to the returned account index.
<i>OkToCache</i>	True if Account Validation Library should cache the result. See Notes for more information.

## Return Values

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

## Error Conditions

HPSS_EUSEDEFAULT	Tells the caller to use the default built-in behavior for this routine.
Other suggested error codes to use are:	
HPSS_EBUSY	The server is busy. The request should be retried later.
HPSS_ENOENT	The account specified does not exist. Acct is set to the integer form of AcctName. If ACCT_FLAGS_VALID_INDEX is set in the Flags parameter, then return HPSS_EPERM instead.
HPSS_EPERM	The specified user is not allowed to use that account name.
HPSS_ESYSTEM	This routine received an internal error. Usually a site customization problem.

## See Also

**av\_AcctNameToIdx**

## Clients

Client API via Gatekeeper

## Notes

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results

are returned and performance is not adversely impacted.

The *OkToCache* argument should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned information can change over time and should not be cached by the client.

### Example Usage

This example is the default behavior of **av\_AcctNameToIdx**. See the implementation of that routine, if available, for more details.

```
av_AcctNameToIdx() {
    lookup_default = (InAcctName is NULL or empty);
    if site style accounting {
        if lookup_default
            read default account for this user from metadata
            if (not found and requiring default account)
                return HPSS_EPERM;
        else if Flags & ACCT_FLAGS_VALID_INDEX
            read this user's account by name
        else
            read account metadata by name
    } else if unix style accounting {
        if (lookup_default)
            *Acct = Uid;
        else {
            try to convert AcctName to number
            if successful, *Acct = number;
        }
    }

    if we set *Acct above {
        error = lookup user from uid
        if not found and require default account
            error = HPSS_EPERM;
    } else {
        error = lookup user from account name
    }
}
```

```

        if (not found and Flags & ACCT_FLAGS_VALID_INDEX
            error = HPSS_EPERM;
    }
    return error;
}

```

### 4.2.3. av\_site\_Initialize

#### Purpose

Customizable routine to perform initialization of the Account Validation APIs.

#### Synopsis

```

signed32
av_site_Initialize(
acct_config_t      *AcctPolicy /* IN */
);

```

#### Description

This routine initializes the site customized APIs which receive client requests. Sites should initialize any common state shared between the various site routines.

#### Parameters

<i>AcctPolicy</i>	Pointer to accounting policy metadata record.
-------------------	---

#### Return Values

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

#### Error Conditions

Suggested error codes to use:

HPSS_SYSTEM	Site code cannot initialize properly.
HPSS_ENOMEM	Out of memory.

#### See Also

**av\_Initialize**

#### Clients

Gatekeeper

#### Notes

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted. Any error returned from this routine is considered fatal and will keep the server from starting up properly.

#### Example Usage



```

av_site_Initialize() {
    allocate space for local site policy information;
    read in local site policy information;
}

```

#### 4.2.4. **av\_site\_Shutdown**

##### **Purpose**

Customizable routine to shutdown the interface to the Account Validation APIs.

##### **Synopsis**

```

signed32
av_site_Shutdown(void) ;

```

##### **Description**

This routine cleans up any state created by the customizable site routines.

##### **Parameters**

None

##### **Return Values**

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

##### **Error Conditions**

Suggested error codes to use:

HPSS_EINVAL	The APIs have not been initialized so there is nothing to shutdown.
-------------	---

##### **See Also**

None

##### **Clients**

Gatekeeper

##### **Notes**

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted. Any error returned from this routine is logged but otherwise ignored.

##### **Example Usage**

```

av_site_Shutdown() {
    deallocate space for local site policy information;
}

```

## 4.2.5. av\_site\_ValidateAccount

### Purpose

Customizable routine to validate that a user is allowed to use an account index.

### Synopsis

```
signed32
av_site_ValidateAccount(
hpss_connect_handle_t  Binding,           /* IN */
hpss_reqid_t           RequestId,         /* IN */
unsigned32              RealmId,          /* IN */
unsigned32              Uid,               /* IN */
acct_rec_t              Acct,             /* IN */
unsigned32              Flags,            /* IN */
acct_rec_t              *ParentAcct,      /* IN */
acct_rec_t              *OutAcct,         /* OUT */
unsigned32              *OkToCache        /* OUT */
);
```

### Description

This routine verifies that Acct is a valid account index to use for the user specified by the Uid and RealmId given. By default, this routine is nothing more than a skeleton that returns HPSS\_EUSEDEFAULT immediately. See av\_ValidateAccount for the default behavior of this routine.

### Parameters

<i>Binding</i>	Binding handle to this server.
<i>RequestId</i>	Request Id to use when logging messages for this request.
<i>RealmId</i>	Realm identifier of user.
<i>Uid</i>	User id (UID) of user.
<i>Acct</i>	Account Index to validate. If this is ACCT_REC_DEFAULT, the specified user's default account index, if any, is used instead.
<i>Flags</i>	One or more of the following OR'd together:  ACCT_VALIDATE_FLAGS_CREATING – Caller is performing a file or directory creation operation.  ACCT_VALIDATE_FLAGS_AUTH_CALLER – Request is being made on the behalf of an authorized caller.
<i>ParentAcct</i>	Pointer to parent account index.
<i>OutAcct</i>	Account Index to use.
<i>OkToCache</i>	True if Account Validation Library should cache the result. See Notes for more information.

### Return Values

Upon successful completion, zero (0) is returned signifying that the account is valid.

A non-zero value indicates an error condition.

## Error Conditions

HPSS\_EUSEDEFAULT      Tells the caller to use the default built-in behavior for this routine.

Other suggested error codes to use are:

HPSS\_EBUSY              The server is busy. The request should be retried later.

HPSS\_EPERM              The specified user is not allowed to use that account index.

HPSS\_ESYSTEM            This routine received an internal error. Usually a site customization problem.

## See Also

**av\_ValidateAccount**

## Clients

Core Server via Gatekeeper

## Notes

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted.

The OkToCache argument should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned information can change over time and should not be cached by the client.

If this request is being performed on the behalf of an authorized caller, it is generally not appropriate to return a permission error from this routine. See **av\_ValidateAccount**.

## Example Usage

This example is the default behavior of **av\_ValidateAccount**. See the implementation of that routine, if available, for more details.

```
av_ValidateAccount() {
    if site style and creating a file and account inheritance is on {
        if (parent account supplied and
            (Acct is ACCT_REC_DEFAULT or parent account)) {
            return with parent account;
        } else if (no parent account supplied and
            Acct != ACCT_REC_DEFAULT) {
            /* assume caller (Core Server) has determined parent already */
            return with Acct;
        }
    }
    if (user is superuser)
```

```

        return with Acct;
    if (user is “nobody”) {
        if auth caller
            return with ACCT_REC_DEFAULT
        return HPSS_EPERM;
    }
    if site style accounting {
        Read the account from metadata.
        If (not found and Acct == ACCT_REC_DEFAULT and auth caller) {
            Create default account for this user;
        } else if not found {
            error = HPSS_EPERM;
        }
    } else if unix style {
        if Acct == UID or ACCT_REC_DEFAULT or auth caller
            return with Uid;
        error = HPSS_EPERM;
    }
    return error;
}

```

## 4.2.6. av\_site\_ValidateChacct

### Purpose

Determine the account code to use when a file’s account index changes.

### Synopsis

```

signed32
av_site_ValidateChacct(
    hpss_connect_handle_t    Binding,           /* IN */
    hpss_reqid_t             RequestId,         /* IN */
    unsigned32               UserRealmId,       /* IN */
    unsigned32               UserUid,           /* IN */
    unsigned32               FileRealmId,       /* IN */
    unsigned32               FileUid,          /* IN */
    unsigned32               FileGid,          /* IN */
    acct_rec_t               OldAcct,          /* IN */
    acct_rec_t               NewAcct,          /* IN */
    acct_rec_t               *OutAcct,         /* OUT */

```

```

    unsigned32          *OkToCache          /* OUT */
);

```

## Description

This routine determines what account index should be used when a file or directory's account index changes. By default, this routine is nothing more than a skeleton that returns HPSS\_EUSEDEFAULT immediately. See **av\_ValidateChacct** for the default behavior of this routine.

## Parameters

<i>Binding</i>	Binding handle to this server.
<i>RequestId</i>	Request Id to use when logging messages for this request.
<i>UserRealmId</i>	Realm identifier of user performing the operation.
<i>UserUid</i>	User id (UID) of user performing the operation.
<i>FileRealmId</i>	Realm identifier of file's owner.
<i>FileUid</i>	User id (UID) of file's owner.
<i>FileGid</i>	Group id (GID) of file's owner.
<i>OldAcct</i>	Old Account Index of file.
<i>NewAcct</i>	New Account Index to put onto the file. If this is ACCT_REC_DEFAULT then the user's default account index is used.
<i>OutAcct</i>	Returned account index to use.
<i>OkToCache</i>	True if Account Validation Library should cache the result. See Notes for more information.

## Return Values

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

## Error Conditions

HPSS_EUSEDEFAULT	Tells the caller to use the default built-in behavior for this routine.
Other suggested error codes to use are:	
HPSS_EBUSY	The server is busy. The request should be retried later.
HPSS_EPERM	The account specified may not be used by this user.
HPSS_ESYSTEM	This routine received an internal error. Usually a site customization problem.

## See Also

**av\_ValidateChacct**

## Clients

Client API via Gatekeeper

## Notes

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted.

The `OkToCache` argument should normally be `TRUE`. If a site customizes this routine it should only set this argument to `FALSE` if the returned information can change over time and should not be cached by the client.

### Example Usage

This example is the default behavior of `av_ValidateChacct`. See the implementation of that routine, if available, for more details.

```
av_ValidateChacct() {
    if user is "nobody", return with *OutAcct = ACCT_REC_DEFAULT;
    if site style accounting {
        if user is superuser and Acct != ACCT_REC_DEFAULT
            return with NewAcct;
        error = read user account from metadata
        if not found {
            if NewAcct == ACCT_REC_DEFAULT, return 0
            else return HPSS_EPERM;
        }
    } else if unix style accounting {
        if NewAcct is FileUid or ACCT_REC_DEFAULT
            return with *OutAcct = FileUid;
        error = HPSS_EPERM;
    }
    return error;
}
```

## 4.2.7. av\_site\_ValidateChown

### Purpose

Determine the account code to use when a file's ownership or group changes.

### Synopsis

```
signed32
av_site_ValidateChown(
hpss_connect_handle_t  Binding,          /* IN */
hpss_reqid_t          RequestId,        /* IN */
unsigned32             OldRealmId,       /* IN */
unsigned32             OldUid,           /* IN */
...)
```

```

unsigned32      OldGid,          /* IN */
acct_rec_t      OldAcct,        /* IN */
unsigned32      NewRealmId,     /* IN */
unsigned32      NewUid,         /* IN */
unsigned32      NewGid,         /* IN */
acct_rec_t      SessionAcct,    /* IN */
acct_rec_t      *OutAcct,       /* OUT */
unsigned32      *OkToCache      /* OUT */
);

```

## Description

This routine determines what account index should be used when a file or directory's ownership changes or its group changes. By default, this routine is nothing more than a skeleton that returns HPSS\_EUSEDEFAULT immediately. See **av\_ValidateChown** for the default behavior of this routine.

## Parameters

<i>Binding</i>	Binding handle to this server.
<i>RequestId</i>	Request Id to use when logging messages for this request.
<i>OldRealmId</i>	Realm identifier of file's owner.
<i>OldUid</i>	User id (UID) of file's owner.
<i>OldGid</i>	Group id (GID) of file.
<i>OldAcct</i>	Account Index of file.
<i>NewRealmId</i>	New Realm identifier of file's owner.
<i>NewUid</i>	New User id (UID) of file's owner.
<i>NewGid</i>	New Group id (GID) of file.
<i>SessionAcct</i>	User's current session account index.
<i>OutAcct</i>	Returned account index to use.
<i>OkToCache</i>	True if Account Validation Library should cache the result. See Notes for more information.

## Return Values

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

## Error Conditions

HPSS_EUSEDEFAULT	Tells the caller to use the default built-in behavior for this routine.
Other suggested error codes to use are:	
HPSS_EBUSY	The server is busy. The request should be retried later.
HPSS_EPERM	The account specified may not be used by this user.
HPSS_ESYSTEM	This routine received an internal error. Usually a site customization problem.

## See Also

## **av\_site\_ValidateChown**

### **Clients**

Client API via Gatekeeper

### **Notes**

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted.

The *OkToCache* argument should normally be TRUE. If a site customizes this routine it should only set this argument to FALSE if the returned information can change over time and should not be cached by the client.

### **Example Usage**

This example is the default behavior of **av\_ValidateChown**. See the implementation of that routine, if available, for more details.

```
av_ValidateChown() {  
    if user is "nobody", return with *OutAcct = ACCT_REC_DEFAULT;  
    if site style accounting {  
        error = read user account from metadata  
        if not found {  
            error = 0;  
            *OutAcct = ACCT_REC_DEFAULT;  
        }  
    } else if unix style accounting {  
        *OutAcct = NewUid;  
    }  
    return error;  
}
```

## **4.2.8. av\_site\_ValidateCreate**

### **Purpose**

Customizable routine to determine the account code to use when creating a file or directory.

### **Synopsis**

```
signed32  
av_site_ValidateCreate (  
    hpss_connect_handle_t    Binding,           /* IN */  
    hpss_reqid_t            RequestId,         /* IN */  
    unsigned32               RealmId,          /* IN */
```



```

unsigned32      Uid,                /* IN */
unsigned32      Gid,                /* IN */
acct_rec_t      Acct,              /* IN */
acct_rec_t      ParentAcct,        /* IN */
acct_rec_t      *OutAcct,          /* OUT */
unsigned32      *OkToCache         /* OUT */
);

```

## Description

This routine determines what account index should be used when a file or directory is created. By default, this routine is nothing more than a skeleton that returns HPSS\_EUSEDEFAULT immediately. See **av\_ValidateCreate** for the default behavior of this routine.

## Parameters

<i>Binding</i>	Binding handle to this server.
<i>RequestId</i>	Request Id to use when logging messages for this request.
<i>RealmId</i>	Realm identifier of new file's owner.
<i>Uid</i>	User id (UID) of new file's owner.
<i>Gid</i>	Group id (GID) of new file.
<i>Acct</i>	Current session account index.
<i>ParentAcct</i>	Account Index of parent directory.
<i>OutAcct</i>	Returned account index to use.
<i>OkToCache</i>	True if Account Validation Library should cache the result. See Notes for more information.

## Return Values

Upon successful completion, zero (0) is returned. A non-zero value indicates an error condition.

## Error Conditions

HPSS_EUSEDEFAULT	Tells the caller to use the default built-in behavior for this routine.
Other suggested error codes to use are:	
HPSS_EBUSY	The server is busy. The request should be retried later.
HPSS_EPERM	The account specified may not be used by this user.
HPSS_ESYSTEM	This routine received an internal error. Usually a site customization problem.

## See Also

**av\_ValidateCreate**

## Clients

Client API via Gatekeeper

## Notes

This routine can be customized by a site. It is the site's responsibility to make sure that consistent results are returned and performance is not adversely impacted.

The `OkToCache` argument should normally be `TRUE`. If a site customizes this routine it should only set this argument to `FALSE` if the returned information can change over time and should not be cached by the client.

### Example Usage

This example is the default behavior of `av_ValidateCreate`. See the implementation of that routine, if available, for more details.

```
av_ValidateCreate() {
    if site style accounting and account inheritance and
        parent account is supplied {
        return with parent account;
    }
    if user is "nobody", return with ACCT_REC_DEFAULT;
    if site style accounting {
        if (user is superuser and Acct != ACCT_REC_DEFAULT)
            return ok with OutAcct = Acct;
        error = read user account from metadata;
        if not found {
            if Acct == ACCT_REC_DEFAULT, return with OutAcct = Acct;
            else return with HPSS_EPERM;
        }
    } else if unix style accounting {
        *OutAcct = Uid;
    }
    log any error;
    return error;
}
```

## Chapter 5. Access Control List API Functions

### 5.1. API Interfaces

This chapter describes all the APIs that are provided in shared object **hacl.so** which is included in the HPSS Client Library **libhpss.a**. In this discussion, the term "object" is used to refer to files and directories. The API interface specification includes the following information:

- Name
- Purpose
- Synopsis
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Notes

#### 5.1.1. hacl\_ConvertACLToHACL

##### Purpose

Convert an access control list to string format.

##### Synopsis

```
#include "hacl.h"
int
hacl_ConvertACLToHACL(
    ns_ACLConfArray_t    *ACL_n,          /* IN */
    hacl_acl_t           **ACL_h);        /* OUT */
```

##### Description

Convert an access control list from the format used by the Client API routines into a format suitable for printing or use by other hacl routines.

##### Parameters

<i>ACL_n</i>	The ACL to be converted
<i>ACL_h</i>	The converted ACL

##### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

##### Error Conditions

EINVAL	ACL_n was not a valid nameserver ACL
--------	--------------------------------------

ENOMEM	Not enough memory available for conversion
(other)	An error returned by <b>hpss_ConvertNamesToIds</b>

#### See Also

**hacl\_ConvertHACLToACL**

#### Notes

When ACL\_h is no longer needed, the memory assigned to it must be returned by calling free().

### 5.1.2. hacl\_ConvertHACLToACL

#### Purpose

Convert an access control list to HPSS format.

#### Synopsis

```
#include "hacl.h"
int
hacl_ConvertHACLToACL(
    hacl_acl_t          *ACL_h,      /* IN */
    ns_ACLConfArray_t  **ACL_n)    /* OUT */
```

#### Description

Convert an access control list into a format suitable for use with the Client API functions that manage access control lists.

#### Parameters

<i>ACL_h</i>	The ACL to be converted
<i>ACL_n</i>	The converted ACL

#### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

#### Error Conditions

EINVAL	ACL_h was not a valid access control list
ENOMEM	Not enough memory available for conversion
(other)	<b>Error returned by hpss_ConvertNamesToIds</b>

#### See Also

**hacl\_ConvertACLToHACL**

#### Notes

When ACL\_n is no longer needed, the memory assigned to it must be returned by calling free().

### 5.1.3. hacl\_ConvertHACLToString

#### Purpose

Convert an access control list to a form suitable for printing

## Synopsis

```
#include "hac1.h"
int
hac1_ConvertHACLToString(
hac1_acl_t      *ACL_h,          /* IN */
int             Flags,          /* IN */
char            *EntrySeparator, /* IN */
char            **ACL_s);       /* OUT */
```

## Description

Convert an access control list into a form suitable for printing.

## Parameters

<i>ACL_h</i>	The ACL to be converted
<i>Flags</i>	Flags that define how output is to be formatted
<i>EntrySeparator</i>	String to be used to separate ACL entries
<i>ACL_s</i>	The converted ACL

## Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

## Error Conditions

ENOMEM	Not enough memory available to hold ACL_S
EINVAL	ACL_h or EntrySeparator or ACL_s is invalid

## See Also

**hpss\_ConvertStringToHACL**

## Notes

The *Flags* argument is the sum of values that determine how the ACL will be formatted. The following bits are defined:

HACL\_M\_USE\_LOCAL\_REALM\_SPEC

HACL\_M\_USE\_MASK\_OBJ

If HACL\_M\_USE\_LOCAL\_REALM\_SPEC is set, the id of the local realm will be displayed in all ACL entries, even those that refer to principals in the local realm. Otherwise the realm will only be displayed for entries that refer to foreign principals.

If HACL\_M\_USE\_MASK\_OBJ is set, the string will be adjusted to reflect the mask object. Otherwise the actual ACL will be shown.

When *ACL\_s* is no longer needed, the memory assigned to it must be returned by calling `free()`.

## 5.1.4. `hacl_ConvertHACLPPermsToPerms`

### Purpose

Convert permission string to HPSS format

### Synopsis

```
#include "hacl.h"
int
hacl_ConvertHACLPPermsToPerms (
char          *HPerms,      /* IN */
unsigned char *Perms);      /* OUT */
```

### Description

Convert a permission string in the format "rwxciid" into a form suitable for use with the Client API ACL management functions. Both uppercase and lower case letters may be used, A hyphen may be used to indicate missing permission bits.

### Parameters

<i>HPerms</i>	The permissions to convert
<i>Perms</i>	The converted permissions

### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

### Error Conditions

EINVAL	Hperms contained an unrecognized character
--------	--

### See Also

`hacl_ConvertPermsToHACLPPerms`

### Notes

None

## 5.1.5. `hacl_ConvertHACLTypeToType`

### Purpose

Convert access control list entry type to HPSS format

### Synopsis

```
#include "hacl.h"
int
hacl_hacl_ConvertHACLTypeToType (
char          HType,        /* IN */
unsigned char *Type);       /* OUT */
```

### Description

Convert an ACL entry type into a form suitable for use with Client API functions. Valid type names

include any of the standard HPSS ACL entry types. These include `user_obj`, `group_obj`, `other_obj`, `user`, `group`, `foreign_user`, `foreign_group`, `foreign_other`, `any_other`, `mask_obj`, `user_obj_delegate`, `group_obj_delegate`, `other_obj_delegate`, `foreign_other_delegate`, and `any_other_delegate`. Strings can be entered in upper or lower case letters. The extended type is not allowed. The unauthorized type is allowed but most Client API routines will treat this ACL entry type as invalid. The output type will be a valid Core Server ACL type such as `ACL_OBJ_MASK`, etc.

#### Parameters

<i>HType</i>	The type to convert
<i>Type</i>	The converted type

#### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

#### Error Conditions

<code>EINVAL</code>	<code>HType</code> is not a valid ACL entry type
---------------------	--

#### See Also

**`hacl_ConvertTypeToHACLType`**

#### Notes

None

## 5.1.6. `hacl_ConvertPermsToHACLPerms`

#### Purpose

Convert HPSS permission mask to string

#### Synopsis

```
#include "hacl.h"
int
hacl_ConvertPermsToHACLPerms (
unsigned char    Perms,      /* IN */
char            HPerms) ;    /* OUT */
```

#### Description

Convert the permissions mask returned by **`hpss_GetACL`** into a form suitable for printing and/or input into other libhacl routines. The output takes the form of a string (e.g., "rwxcid") with missing permissions replaced by a hyphen (e.g., "rwx---").

#### Parameters

<i>Perms</i>	The permissions to convert
<i>HPerms</i>	The converted permissions

#### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

## Error Conditions

EINVAL	Perms is not a valid permission mask
--------	--------------------------------------

## See Also

**hacl\_ConvertHACLPermsToPerms**

## Notes

None

## 5.1.7. hacl\_ConvertStringsToHACL

### Purpose

Convert access control list strings to HACL format

### Synopsis

```
#include "hacl.h"
int
hacl_ConvertStringsToHACL(
int          NumEntries,          /* IN */
char         *Entries[],          /* IN */
char         *DefaultRealm,       /* IN */
char         *WhiteSpaceChars,    /* IN */
int          Flags,               /* IN */
hacl_acl_t   **ACL_h);           /* OUT */
```

### Description

Convert one or more string representations of access control list entries into a structure for input to other libhacl functions.

### Parameters

<i>NumEntries</i>	Number of elements in Entries
<i>Entries</i>	Array of ACL strings
<i>DefaultRealm</i>	Default Realm
<i>WhiteSpaceChars</i>	Entry separator(s)
<i>Flags</i>	Flags
<i>ACL_h</i>	Resulting ACL

### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

### Error Conditions

ENOMEM	Cannot allocate memory for ACL
EINVAL	One or more ACL entries is in error
ENAMETOOLONG	The length of DefaultRealm exceeds a system-imposed limit



## See Also

**hacl\_ConvertHACLToString**

## Notes

The Entries argument is an array of strings, each of which represents one or more ACL entries. If there are several ACLs in any one string, they are separated by the character(s) listed in WhiteSpace. The Flags argument defines the format of the incoming string. The following bits are defined:

HACL\_M\_REQUIRE\_PERMS

HACL\_M\_USE\_LOCAL\_REALM\_SPEC

If HACL\_M\_REQUIRE\_PERMS is set, then the permission string must be supplied as part of the ACL entry. Otherwise, the permission string is optional. The first case would be used when an ACL is being created, while the second case would be used when an ACL entry is being deleted.

If HACL\_M\_USE\_LOCAL\_REALM\_SPEC is set, the id of the local realm can optionally be provided in all ACL entries, even those that refer to principals in the local realm. Otherwise the realm should be specified only for entries that refer to foreign principals.

When ACL\_h is no longer needed, the memory assigned to it must be returned using free().

## 5.1.8. hacl\_ConvertTypeToHACLType

### Purpose

Convert an access control list entry type to string format

### Synopsis

```
#include "hacl.h"
int
hacl_ConvertTypeToHACLType (
    unsigned char    Type,          /* IN */
    char             HType);        /* OUT */
```

### Description

Convert an ACL entry type into a form suitable for printing or use with other functions in libhacl. The input type should be a valid nameserver ACL type such as ACL\_OBJ\_MASK, etc. The output is the corresponding string (eg, "mask\_obj").

### Parameters

<i>Type</i>	The entry type to convert
<i>HType</i>	The converted type

### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

### Error Conditions

EINVAL	Type was not a valid ACL entry type
--------	-------------------------------------

## See Also

## hacl\_ConvertHACLTypeToType

### Notes

None

## 5.1.9. hacl\_DeleteHACL

### Purpose

Delete selected entries from an object's access control list

### Synopsis

```
#include "hacl.h"
int
hacl_DeleteHACL(
char          *Path,          /* IN */
unsigned32    Options,        /* IN */
hacl_acl_t    *ACL_h);       /* IN */
```

### Description

Delete the access control list entries specified in ACL\_h from the file named by Path. The Options argument is used to select whether an initial container or initial object ACLs are to be processed.

### Parameters

<i>Path</i>	Path to the object
<i>Options</i>	Processing options
<i>ACL_h</i>	ACL entries to be deleted

### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

### Error Conditions

EACCES	Search permission is denied on a component in Path
EFAULT	The Path or ACL parameter is a NULL pointer
EINVAL	ACL_h contains invalid data
ENAMETOOLONG	The length of Path exceeds a system-imposed limit
ENOENT	The named object does not exist, or Path is an empty string
ENOTDIR	A component of Path is not a directory
EPERM	No privilege for attempted operation
ESRCH	A specified ACL entry could not be found in the object's existing ACL

### See Also

**hacl\_GetHACL**

### Notes

The Options flag determines processing options. Values include:

HPSS\_ACL\_OBJECT\_ACL - regular ACL

HPSS\_ACL\_INITIAL\_CONTAINER\_ACL - initial container ACL

HPSS\_ACL\_INITIAL\_OBJECT\_ACL - initial object ACL

## 5.1.10. hacl\_GetHACL

### Purpose

Get an object's access control list

### Synopsis

```
#include "hacl.h"
int
hacl_GetHACL(
    char          *Path,          /* IN */
    unsigned32    Options,        /* IN */
    hacl_acl_t    **ACL_h);      /* OUT */
```

### Description

Get the access control list of a object. The Options argument is used to select whether an initial container or initial object ACL is to be obtained.

### Parameters

<i>Path</i>	Path to the object
<i>Options</i>	Processing options
<i>ACL_h</i>	The object's access control list

### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

### Error Conditions

EACCES	Search permission is denied on a component in Path
EFAULT	The Path or ACL parameter is a NULL pointer
EINVAL	Path or Acl_h is invalid
ENAMETOOLONG	The length of Path exceeds a system-imposed limit
ENOENT	The named object does not exist, or Path is an empty string
ENOTDIR	A component of Path is not a directory
EPERM	No privilege for attempted operation

### See Also

**hacl\_SetHACL, hacl\_DeleteHACL**

### Notes

The Options flag determines processing options. Values include:

HPSS_ACL_OBJECT_ACL	The Path or ACL parameter is a NULL pointer
HPSS_ACL_INITIAL_CONTAINER_ACL	Initial container ACL
HPSS_ACL_INITIAL_OBJECT_ACL	Initial object ACL

When ACL\_h is no longer needed, the memory assigned to it must be returned by calling free().

## 5.1.11. hacl\_SetHACL

### Purpose

Replace an object's access control list with a new one

### Synopsis

```
#include "hacl.h"
int
hacl_SetHACL(
    char          *Path,          /* IN */
    unsigned32    Options,       /* IN */
    hacl_acl_t    *ACL_h);      /* IN */
```

### Description

Replace the access control list of the object named by Path with a new one given by ACL\_h. The Options argument is used to select whether an initial container or initial object ACL are to be processed.

### Parameters

<i>Path</i>	Path to the object
<i>Options</i>	Processing options
<i>ACL_h</i>	The new ACL

### Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

### Error Conditions

EACCES	Search permission is denied on a component in Path
EFAULT	The Path or ACL parameter is a NULL pointer
EINVAL	ACL_h contains invalid data
ENAMETOOLONG	The length of Path exceeds a system-imposed limit
ENOENT	The named object does not exist, or Path is an empty string
ENOTDIR	A component of Path is not a directory
EPERM	No privilege for attempted operation

### See Also

**hacl\_GetHACL, hacl\_DeleteHACL**

## Notes

The Options flag determines processing options. Values include:

HPSS_ACL_OBJECT_ACL	Regular ACL
HPSS_ACL_INITIAL_CONTAINER_ACL	Initial container ACL
HPSS_ACL_INITIAL_OBJECT_ACL	Initial object ACL

## 5.1.12. `hacl_SortHACL`

### Purpose

Sort an access control list into canonical order

### Synopsis

```
#include "hacl.h"
void
hacl_SortHACL(
    hacl_acl_t      *ACL_h);    /* IN/OUT */
```

### Description

Sort the entries of an access control list into a standard order.

### Parameters

*ACL\_h*                      The ACL to be sorted

### Return Values

None.

### Error Conditions

None. The function always succeeds.

### See Also

None

## Notes

The "standard" order arranges ACL entries in the order that they will be considered to decide whether a principal has access to an object. For example, the `mask_obj` will appear first, followed by the `user_obj`, `user`, `foreign_user`, etc. entries. If there are two entries of the same type, they will be sorted based on the `EntryName` and/or `RealmName` fields.

This routine should be called before displaying an ACL using `hacl_ConvertHACLToString`.

## 5.1.13. `hacl_UpdateHACL`

### Purpose

Change selected entries in an object's access control list

## Synopsis

```
#include "hacl.h"
int
hacl_UpdateHACL(
    char          *Path,          /* IN */
    unsigned32    Options,        /* IN */
    hacl_acl_t    *ACL_h);       /* IN */
```

## Description

Update selected entries in an access control list of an object. The Options argument is used to select whether an initial container or initial object ACL is to be processed.

## Parameters

<i>Path</i>	Path to the object
<i>Options</i>	Processing options
<i>ACL_h</i>	ACL to be updated

## Return Values

If successful, returns zero. Otherwise, returns one of the values described below.

## Error Conditions

EACCES	Search permission is denied on a component in Path
EFAULT	The Path or ACL parameter is a NULL pointer
EINVAL	ACL_h contains invalid data
ENAMETOOLONG	The length of Path exceeds a system-imposed limit
ENOENT	The named object does not exist, or Path is an empty string
ENOTDIR	A component of Path is not a directory
EPERM	No privilege for attempted operation

## See Also

**hacl\_DeleteACL, hacl\_GetACL, hacl\_SetACL**

## Notes

The Options flag determines processing options. Values include:

HPSS_ACL_OBJECT_ACL	Regular ACL
HPSS_ACL_INITIAL_CONTAINER_ACL	Initial container ACL
HPSS_ACL_INITIAL_OBJECT_ACL	Initial object ACL
HPSS_ACL_DONT_CALC_MASK	Don't calculate mask
HPSS_ACL_CALC_MASK_IGNORE_ERRORS	Calculate mask unconditionally

## 5.2. Data Definitions

This section describes the data definitions used by the routines in libhac1.

### 5.2.1. HAC1-style Access Control List - hac1\_acl\_t

#### Description

The Access Control List structure defines an array of ACL entries.

#### Format

```
typedef struct hac1_acl {  
    size_t      Length;  
    char        DefaultRealm[HPSS_MAX_REALM_NAME];  
    hac1_acl_entry_t ACLEntry[1];  
} hac1_acl_t;
```

#### Length

The number of ACL entries in array ACLEntry.

#### DefaultRealm

The realm used to set a context for interpreting the ACL.

#### ACLEntry

An array of ACL entries.

### 5.2.2. HAC1-style Access Control List Entry - hac1\_acl\_entry\_t

#### Description

The Access Control List Entry structure defines one entry in an ACL.

#### Format

```
typedef struct hac1_acl_entry {  
    char  EntryType[HACL_MAX_TYPE];  
    char  Perms[HACL_MAX_PERMS];  
    char  EntryName[HPSS_MAX_PRINCIPAL_NAME];  
    char  RealmName[HPSS_MAX_REALM_NAME];  
} hac1_acl_entry_t;
```

#### EntryType

The type of ACL entry; e.g., "user\_obj", etc.

#### Perms

The permissions, composed of characters in "rwxci-".

#### EntryName

Depending on the EntryType, the User name, group name, or realm name.

#### RealmName

The realm name. Qualifies user and group names to make them unique.



## Chapter 6. HPSS File System Client Interfaces

This chapter describes the non-POSIX interfaces for the HPSS File System Client. The interfaces are in the form of `ioctl(2)` commands that can be used to control the HPSS specific properties for files being accessed via the File System Client interface. The interface specification includes the following information:

- Name
- Purpose
- Synopsis
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Clients
- Notes
- Example Usage

### 6.1.1. HPSSFS\_GET\_COS

#### Purpose

Get the HPSS COS identifier for a file

#### Synopsis

```
#include <linux/hpssfs.h>

int
ioctl(
    int          fd,          /* IN - Open file descriptor */
    int          cmd,          /* IN - Command (HPSSFS_GET_COS) */
    unsigned int *cosid);      /* OUT - HPSS COS identifier */
```

#### Description

This function is used to obtain the HPSS COS identifier for the open file, specified by the file descriptor, *fd*.

#### Parameters

<i>fd</i>	Open file descriptor
<i>cmd</i>	Command to execute (i.e. HPSSFS_GET_COS)
<i>cosid</i>	Pointer to the returned HPSS COS identifier for the file

#### Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

## Error Conditions

The `ioctl(HPSSFS_GET_COS)` routine is unsuccessful if any of the following are true:

ENOTTY	File descriptor is not associated with an HPSS file
EFAULT	Pointer value is invalid

## See Also

**HPSSFS\_SET\_COS\_HINT**

## Clients

HPSS File System clients

## Notes

None.

## Example Usage

Open the HPSS file, with the `open(2)` system call.

Issue the the `HPSS_GET_COS ioctl(2)` system call.

## 6.1.2. HPSSFS\_SET\_COS\_HINT

### Purpose

Set the HPSS COS identifier hint for a file

### Synopsis

```
#include <linux/hpssfs.h>

int
ioctl(
    int          fd,          /* IN - Open file descriptor */
    int          cmd,          /* IN - Command (HPSSFS_SET_COS_HINT) */
    unsigned int *cosid);      /* IN - HPSS COS identifier */
```

### Description

This function is used to set the HPSS COS identifier hint for the open file, specified by the file descriptor, *fd*. Setting this hint, before any data is written to the file, allows HPSS to store file data in the corresponding HPSS COS.

### Parameters

<i>fd</i>	Open file descriptor
<i>cmd</i>	Command to execute (i.e. <code>HPSSFS_SET_COS_HINT</code> )
<i>cosid</i>	Pointer to the returned HPSS COS identifier for the file

### Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

## Error Conditions

The **ioctl(HPSSFS\_SET\_COS\_HINT)** routine is unsuccessful if any of the following are true:

ENOTTY	File descriptor is not associated with an HPSS file
EFAULT	Pointer value is invalid
EBADF	File descriptor was not obtained in write mode
EPERM	Data length for the file is not zero

## See Also

**HPSSFS\_SET\_FSIZE\_HINT** and **HPSSFS\_SET\_MAXSEGSZ\_HINT**

## Clients

HPSS File System clients

## Notes

The file descriptor must be opened in write mode (i.e. O\_WRONLY or O\_RDWR).

## Example Usage

Create the HPSS file, with the open(2) or create(2) system call.

Issue the the HPSS\_SET\_COS\_HINT ioctl(2) system call.

Write data to the file.

## 6.1.3. HPSSFS\_SET\_FSIZE\_HINT

### Purpose

Set the HPSS file size hint for a file

### Synopsis

```
#include <linux/hpssfs.h>

int
ioctl(
    int          fd,          /* IN - Open file descriptor */
    int          cmd,         /* IN - Command (HPSSFS_SET_FSIZE_HINT) */
    off_t        *size);     /* IN - File size */
```

### Description

This function is used to set the HPSS file size hint for the open file, specified by the file descriptor, *fd*. Setting this hint, before any data is written to the file, allows HPSS to select the segment size that best fits the specified file size.

### Parameters

<i>fd</i>	Open file descriptor
<i>cmd</i>	Command to execute (i.e. HPSSFS_SET_FSIZE_HINT)

*size*                                      Pointer to the size in bytes for the file

### Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

### Error Conditions

The **ioctl(HPSSFS\_SET\_FSIZE\_HINT)** routine is unsuccessful if any of the following are true:

ENOTTY	File descriptor is not associated with an HPSS file
EFAULT	Pointer value is invalid
EBADF	File descriptor was not obtained in write mode
EPERM	Data length for the file is not zero

### See Also

**HPSSFS\_SET\_COS\_HINT** and **HPSSFS\_SET\_MAXSEGSZ\_HINT**

### Clients

HPSS File System clients

### Notes

The file descriptor must be opened in write mode (i.e. O\_WRONLY or O\_RDWR).

### Example Usage

Create the HPSS file, with the `open(2)` or `create(2)` system call.

Issue the the `HPSSFS_SET_FSIZE_HINT` `ioctl(2)` system call.

Write data to the file.

## 6.1.4. HPSSFS\_SET\_MAXSEGSZ\_HINT

### Purpose

Enable maximum segment size selection hint for a file

### Synopsis

```
#include <linux/hpssfs.h>

int
ioctl(
    int          fd,          /* IN - Open file descriptor */
    int          cmd,         /* IN - Command (HPSSFS_SET_MAXSEGSZ_HINT) */
    off_t        *setmax);   /* IN - Max segment enabler */
```

### Description

This function is used to enable maximum segment size selection hint for the open file, specified by the file descriptor, *fd*. Setting this hint, before any data is written to the file, allows HPSS to select the maximum segment size value in the associated COS, for segment creation.

## Parameters

<i>fd</i>	Open file descriptor
<i>cmd</i>	Command to execute (i.e. HPSSFS_SET_MAXSEGSZ_HINT)
<i>setmax</i>	Pointer to a boolean value that indicates that maximum segment selection is enabled (e.g. 0 => no max segment, ~0 => max segments)

## Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

## Error Conditions

The `ioctl(HPSSFS_SET_MAXSEGSZ_HINT)` routine is unsuccessful if any of the following are true:

ENOTTY	File descriptor is not associated with an HPSS file
EFAULT	Pointer value is invalid
EBADF	File descriptor was not obtained in write mode
EPERM	Data length for the file is not zero

## See Also

`HPSSFS_SET_COS_HINT` and `HPSSFS_SET_FSIZE_HINT`

## Clients

HPSS File System clients

## Notes

The file descriptor must be opened in write mode (i.e. `O_WRONLY` or `O_RDWR`).

## Example Usage

Create the HPSS file, with the `open(2)` or `create(2)` system call.

Issue the `HPSSFS_SET_MAXSEGSZ_HINT` `ioctl(2)` system call.

Write data to the file.

## 6.1.5. HPSSFS\_PURGE\_CACHE

### Purpose

Purge cached data for the file

### Synopsis

```
#include <linux/hpssfs.h>

int
ioctl(
    int          fd,          /* IN - Open file descriptor */
    int          cmd,         /* IN - Command (HPSSFS_PURGE_CACHE) */
    off_t        *purge);    /* IN - Purge flag */
```

## Description

This function is used to purge all data from the page cache for the open file, specified by the file descriptor, *fd*. Subsequent file access will require the data to be read from HPSS disks.

## Parameters

<i>fd</i>	Open file descriptor
<i>cmd</i>	Command to execute (i.e. HPSSFS_PURGE_CACHE)
<i>purge</i>	Pointer to a boolean value that indicates that the cache is to be purged (e.g. 0 => no purge of cache, ~0 => purge cache)

## Return Values

Upon successful completion, a value of zero (0) is returned. Non-zero values are described in the Error Conditions.

## Error Conditions

The **ioctl(HPSSFS\_PURGE\_CACHE)** routine is unsuccessful if any of the following are true:

ENOTTY	File descriptor is not associated with an HPSS file
EFAULT	Pointer value is invalid

## See Also

None.

## Clients

HPSS File System clients

## Notes

None.

## Example Usage

Open the HPSS file, with the open(2) system call.

Issue the the HPSS\_PURGE\_DATA ioctl(2) system call.

## Chapter 7. Configuration Parser Functions

This chapter specifies the HPSS configuration parser interface. The configuration parser is used to parse the contents of the **HPSS.conf** file, and the user should refer to the **HPSS.conf** man pages for the syntax and terminology. Specifically, the following information is provided here:

- Application Programming Interfaces (APIs)
- Data Definitions

### Configuration Parser Interfaces

This section describes all configuration parser interfaces which are provided for use by another HPSS subsystem or by a client external to HPSS. The configuration parser interface specification includes the following information:

- Name
- Purpose
- Synopsis
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Notes

The Configuration Parser has been rewritten for HPSS 6.2.

#### 7.1.1. hpsscfx\_ConfGetClientInterfaces

##### Purpose

Lookup client interfaces by host

##### Synopsis

```
int
hpsscfx_ConfGetClientInterfaces(
    char        *hostName,      /* IN */
    char        *serverName,    /* IN */
    int         *nwIfCount,     /* OUT */
    struct in_addr **nwIfList,   /* OUT */
    char        *nwIfNames);    /* OUT */
```

##### Description

This function looks up a list of client interfaces that match the supplied hostname (if non-NULL)

##### Parameters

<i>hostName</i>	Client Hostname to use for the search (Null is local host)
<i>serverName</i>	Remote host to use for the search

<i>nwIfCount</i>	The number of Interfaces found
<i>nwIfList</i>	A malloced list of the found client Network Addresses (132.111.13.1)
<i>nwIfNames</i>	A malloced list of the found client Interface Names (eth0)

### Return Values

Success (0) or error code.

### Error Conditions

<i>errno</i>	System error code.
HPSS_ENOENT	No entries found

### See Also

None.

### Notes

None.

## 7.1.2. hpss\_CfgParse

### Purpose

Build the internal configuration table.

### Synopsis

```
int
hpss_CfgParse (
    char          *cfgPath,    /* IN */
    hpss_cfg_stanza_t **Head,  /* OUT */
    int           *parseErr,   /* OUT */
    int           *errLine);  /* OUT */
```

### Description

This function is called to parse a configuration file containing stanzas of the form defined for the **HPSS.conf** file. See the comments in **hpss\_config\_api.h** for details of the file structure.

If the file is parsed successfully, then the head of the linked list of stanzas is returned to the caller in **Head**. On success, the number of lines in the configuration file is returned in **errLine**; if a fatal parse error occurs, **errLine** is set to the line number (starting at 1) on which the error was detected. **parseErr** contains the specific error code.

### Parameters

<i>cfgPath</i>	Pathname to the Configuration file name
<i>Head</i>	Pointer to the head of the parsed entries.
<i>ParseErr</i>	Pointer to the parse error code
<i>errLine</i>	Pointer to the line number containing the error.

### Return Values



0 or -1

## Error Conditions

<code>parseErr</code>	Points to internal parsing error values
<code>errLine</code>	Points to the number of lines in the file or the line in the file where the error was observed. See <b>hpss_config_api.h</b> for definitions.
<code>HPSS_EINVAL</code>	More than one configuration table open at one time.

## See Also

None.

## Notes

The caller is responsible for calling **!ksh** to free up memory for the list when it is no longer needed.

## 7.1.3. hpss\_CfgFindToken

### Purpose

Find a token within a key.

### Synopsis

```
hpss_cfg_stanza_t *  
hpss_CfgFindToken(  
    char                *searchToken,           /* IN */  
    hpss_cfg_stanza_t *cfgStart),              /* IN */  
    int                 keyOrValue,             /* IN */  
    int                 subLevels);             /* IN */
```

### Description

This function is called to search for a white-space separated token within a key for a particular level.

### Parameters

<i>searchToken</i>	Token for which to search.
<i>cfgStart</i>	Pointer to starting Location.
<i>keyOrValue</i>	Search Criteria (Key or Value)
<i>subLevels</i>	Search this level ONLY or recursively.

### Return Values

NULL	Token NOT found
Non-NULL	Pointer to the entry.

## Error Conditions

None

## See Also

None

#### Notes

None

### 7.1.4. hpss\_CfgFindKey

#### Purpose

Search for a Specific Key

#### Synopsis

```
hpss_cfg_stanza_t  
hpss_CfgFindKey(  
    char                *searchStr,          /* IN */  
    hpss_cfg_stanza_t *cfgStart),           /* IN */  
    int                 caseFlag);          /* IN */
```

#### Description

This function is called to search for a Key value for a particular level. If caseFlag is 0, then the search is case-sensitive, otherwise, the search is case-insensitive.

#### Parameters

<i>searchStr</i>	Token for which to search.
<i>cfgStart</i>	Pointer to starting Location.
<i>caseFlag</i>	Search Criteria (Sensitive / Non-Sensitive)

#### Return Values

NULL	Key NOT found
Non-NULL	Pointer to the entry.

#### Error Conditions

None

#### See Also

None

#### Notes

None

### 7.1.5. hpssFree

#### Purpose

Find the specified hostname.

#### Synopsis

```
void
```

```

hpss_CfgFree (
    hpss_cfg_stanza_t    *Head);           /* IN */

```

### Description

Recursively free all the linked lists startint Head

### Parameters

*Head*                      Pointer to the start of the list.

### Return Values

None

### Error Conditions

None

### See Also

None

### Notes

None

## 7.1.6. hpsscfx\_LookupHostName

### Purpose

Find the specified hostname.

### Synopsis

```

hpss_cfg_stanza_t *
hpsscfx_LookupHostName (
    hpss_cfg_stanza_t    *hostStanzaList,    /* IN */
    char                  *hostName,          /* IN */
    int                    ignoreDomain);     /* IN */

```

### Description

This function attempts to find a substanza for the hostname contained in the **hostName** parameter in the linked list of hostnames whose head is pointed to by the **hostStanzaList** parameter.

The name is searched for literally, i.e., if it contains a domain name, then the client host entry must also contain a domain name or a wildcard pattern in order to match, except that if the **ignoreDomain** flag is TRUE, then any domain name is first stripped from the passed-in hostname.

The client host list contains one or space-separated names for the KeyString member of the hpss\_cfg\_stanza\_t structure. Each of these is treated as a pattern, which can contain zero or more wildcard characters. For example, the entry:

```

mda-? mda-?.sdsc.edu = {
    ...
}

```

would match any of the following hostnames:

mda-1 mda-7 mda-3.sdsc.edu

If the **ignoreDomain** parameter is TRUE, then the domain name part of **hostName** (if included as part of the name) is ignored.

#### Parameters

<i>hostStanzaList</i>	Pointer to the start of the list.
<i>hostName</i>	Host name for which to search.
<i>ignoreDomain</i>	Flag specifying whether or not the Domain Name should be ignored.

#### Return Values

NULL	Entry not found.
Non-NULL	Pointer to entry

#### Error Conditions

None

#### See Also

None

#### Notes

None

### 7.1.7. hpsscfx\_ConfParse

#### Purpose

Parse a configuration file

#### Synopsis

```
hpss_cfg_stanza_t *  
hpsscfx_ConfParse(void);
```

#### Description

This function is called to parse a configuration file, if it can be found, creating an internal "hpss\_CfgEntries" table. The configuration file is looked for in a number of places, in the following order:

1. The Configuration file directory specified in the HPSS\_CFG\_FILE\_PATH environment variable.
2. If **hpsscfx\_ConfSetPathDirs** was called prior to this call, the paths to the configuration file to be parsed is contained in the <hpssConfPathDir> global variable.
3. In /usr/local/etc
4. In /var/hpss/etc

For cases 2 – 4, the name of the Configuration File will be **HPSS.conf** file unless the `hpss_ConfSetFileName()` routine has been called prior to calling this routine.

If the `hpss_CfgEntries` table has already been built, then this call does nothing. It is necessary to free and NULL `hpss_CfgEntries` to perform this routine more than one time.

### Parameters

None.

### Return Values

NULL	Configuration file not found.
Non-NULL	Pointer to the parsed table.

### Error Conditions

None

### See Also

None

### Notes

None

## 7.1.8. hpsscfx\_ConfSetPathDirs

### Purpose

Set search path(s) for the configuration files.

### Synopsis

```
int
hpsscfx_ConfSetPathDirs (
    char                *thePathString)        /* IN */
```

### Description

This function provides an API to allow the application to supply the path(s) to be searched for the configuration file. These paths will be checked first when the file is parsed (in function **hpsscfx\_ConfParse**) unless the HPSS\_CFG\_FILE\_PATH environment variable is set.

### Parameters

<i>thePathString</i>	The path(s) to be searched. Multiple paths may be specified separated by a colon ( : ), e.g. /usr/local/etc:/var/hpss/etc
----------------------	---

### Return Values

0	Success.
---	----------

### Error Conditions

HPSS_EFAULT	Null pathname passed in.
HPSS_ENOMEM	Memory allocation error.

### See Also

None

## Notes

1. If multiple calls are made, the last one overrides.
2. The colon character cannot be part of the path.

## 7.1.9. hpsscfx\_ConfSetFileName

### Purpose

Set the name of the configuration file.

### Synopsis

```
int
hpsscfx_ConfSetFileName (
    char                *theFileName)    /* IN */
```

### Description

This function provides an API to allow the application to supply an alternate configuration filename to be used. The default will be **HPSS.conf**.

### Parameters

<i>theFileName</i>	The name of the configuration file.
--------------------	-------------------------------------

### Return Values

0	Success.
---	----------

### Error Conditions

HPSS_EFAULT	Null pathname passed in.
HPSS_ENOMEM	Memory allocation error.

### See Also

None

## Notes

1. If multiple calls are made, the last one overrides.

## 7.1.10. hpsscfx\_ConfSetDirPaths

### Purpose

Set the name of the configuration file.

### Synopsis

```
int
hpsscfx_ConfSetDirPaths (
    char                *theDirName)    /* IN */
```

### Description

This function provides an API to allow the application to supply an alternate directory path to be used. The

default is specified by the value of the HPSS\_CFG\_FILE\_PATH environment variable, /usr/local/etc, or /var/hpss/etc in that order.

#### Parameters

<i>theDirName</i>	The name of the path to the configuration file.
-------------------	---

#### Return Values

0	Success.
---	----------

#### Error Conditions

HPSS_EFAULT	Null pathname passed in.
HPSS_ENOMEM	Memory allocation error.

#### See Also

None

#### Notes

None

## 7.1.11. hpsscfx\_NetoptFindEntry

#### Purpose

Searches through the network option table, if present, to find network options configured for the specified address.

#### Synopsis

```
int
hpsscfx_NetoptFindEntry (
    caddr_t      *NetAddr,      /* IN */
    netopt_entry_t **RetEntryPtr) /* OUT */
```

#### Description

Rumbles through the network option table, attempting to initialize the table if not already done, to find an entry that matches the specified address - returning a pointer to the table entry.

If a Default host entry was defined in the Network Options section of the HPSS.conf file, then it will always be returned if no other matching entry is found. If multiple Default entries were mistakenly configured, the first such entry will be used.

#### Parameters

<i>NetAddr</i>	Address to be searched for.
<i>RetEntryPtr</i>	Pointer to table entry.
<i>IgnoreDomain</i>	Flag specifying whether or not the Domain Name should be ignored.

#### Return Values

0	Success.
-1	No Entry Found

#### Error Conditions

None

#### See Also

None

#### Notes

None

## 7.1.12. hpsscfx\_NetoptSetSock

#### Purpose

Using either a network address passed as the second parameter or an address determined from the file descriptor first parameter, set the socket options for the file descriptor and set the global NetOpt variables.

#### Synopsis

```
int
hpsscfx_NetoptSetSock (
    int          *Fd,          /* IN */
    struct sockaddr *NetAddress); /* IN */
```

#### Description

First determine the remote network address from first NetAddress or second from the connected socket. Lookup up the network options. Use setsockopt to set the network options.

#### Parameters

<i>Fd</i>	File descriptor.
<i>NetAddress</i>	Network address.

#### Return Values

0	Success.
-1	No Entry Found

#### Error Conditions

None

#### See Also

None

#### Notes

None



## 7.1.13. hpsscfx\_NetoptGetWriteSize

### Purpose

Return the network (TCP/IP) write size to be used for the specified connection.

### Synopsis

```
int
hpsscfx_NetoptGetWriteSize(
    int          SocketDescriptor,    /* IN */
    long         IpAddr);             /* IN */
```

### Description

Attempts to find a network option entry for the specified connection (based on the local address) - if found and the entry contains a non-zero value for the network write size, that value is returned. Otherwise, the default value (based on the presence of the "HPSS\_TCP\_WRITESIZE" environment variable) is returned. If the *IpAddr* is 0 on entry, then it is determined by getting the peer address for the socket.

### Parameters

<i>SocketDescriptor</i>	Connection file descriptor.
<i>IpAddr</i>	IP address of the connection

### Return Values

Size	Size to be used.
------	------------------

### Error Conditions

None
------

### See Also

None
------

### Notes

None
------

## 7.1.14. hpsscfx\_PatternMatch

### Purpose

Pattern match function.

### Synopsis

```
int
hpsscfx_PatternMatch(
    char         *theString,          /* IN */
    char         *thePattern,         /* IN */
    int          *patternError);      /* OUT */
```

### Description

This function implements CSH-style pattern matching for a candidate string (*theString*) and a pattern (*thePattern*). See the prologue to "matchPattern" for a description of the pattern matching. If an error is encountered on the pattern string, then *\*patternError* is set non-zero.

#### Parameters

<i>theString</i>	String to test for a pattern match.
<i>thePattern</i>	Pattern to compare against.
<i>patternError</i>	Set non-zero on exit if pattern error.

#### Return Values

TRUE	Match found.
FALSE	No match found.

#### Error Conditions

None

#### See Also

None

#### Notes

None

## 7.1.15. hpsscfx\_GetRestrictedPorts

#### Purpose

Read and set the restricted ports

#### Synopsis

```
void
hpsscfx_GetRestrictedPorts (
    static int    minPort, /* OUT */
    static int    maxPort) /* OUT */
```

#### Description

This function is called to get the range of restricted ports that are allowed, based upon the environment variables:

RPC_RESTRICTED_PORTS	Set to restrict which ports are used for communication.
HPSS_PFTPC_PORT_RANGE	When RPC_RESTRICTED_PORTS is not used, use this environment variable to restrict the ports used for communication. This is for compatibility with PFTP.

RPC\_RESTRICTED\_PORTS takes precedence, if both are specified.

If no ports are restricted, the function returns:

minPort = 0

maxPort = 65535

#### Parameters

<i>minPort</i>	The minimum port range value.
<i>maxPort</i>	The maximum port range value.

#### Return Values

None

#### Error Conditions

None

#### See Also

None

#### Notes

None

Architecture Independent (ai) thread, mutex, condition routines – require platform specifications in **ai\_thread\_defs.h**

## 7.1.16. ai\_thread\_abort

#### Purpose

Abort or cancel a thread.

#### Synopsis

```
int
ai_thread_abort(
    ai_thread_t theThread);    /* IN */
```

#### Description

This function initiates the abort or cancel of a thread.

#### Parameters

<i>theThread</i>	Thread to be terminated.
------------------	--------------------------

#### Return Values

Success (0) or error code.

#### Error Conditions

errno	System error code.
-------	--------------------

#### See Also

None.

#### Notes

None

### 7.1.17. ai\_thread\_detach

#### Purpose

Detach a thread.

#### Synopsis

```
int
ai_thread_detach(
    ai_thread_t theThread);          /* IN */
```

#### Description

This function is called to indicate that ai\_thread\_join will never be called on the given thread.

#### Parameters

*theThread*                      Thread ID to be detached.

#### Return Values

Success (0) or error code.

#### Error Conditions

errno                          System error code.

#### See Also

None.

#### Notes

None.

### 7.1.18. ai\_thread\_exit

#### Purpose

Exit a thread

#### Synopsis

```
void
ai_thread_exit(
    void *Status);                  /* OUT */
```

#### Description

This function terminates the calling thread. The result is passed to the thread that joins the caller, or is discarded if the caller is detached.

An implicit ai\_thread\_exit occurs when the top-level (thread start) function returns [PTHREADS, POSIX\_PTHREADS]

#### Parameters

*Status* Exit status.

#### Return Values

Success (0) or error code.

#### Error Conditions

*errno* System error code.

#### See Also

None.

#### Notes

None.

## 7.1.19. ai\_thread\_create

#### Purpose

Create a new thread

#### Synopsis

```
int
ai_thread_exit(
    ai_thread_t      *theThread,
    ai_thread_attr_t  threadAttr,
    void             (*_Start_routine)(void *),
    void             *Arg);
```

#### Description

This function is used to start up a new thread. The attributes structure for the new thread may be passed in the threadAttr variable, or the constant value ai\_thread\_attr\_default may be used to specify default settings.

#### Parameters

*theThread* Thread to be started.

*threadAttr* Thread attributes.

*\*(\_Start\_routine)(void \*)* Pointer to the thread.

*Arg* Arguments for the thread.

#### Return Values

Success (0) or error code.

#### Error Conditions

*errno* System error code.

#### See Also

None.

## Notes

None.

## 7.1.20. ai\_thread\_equal

### Purpose

Check if two threads have the same thread identified (They are the same thread).

### Synopsis

```
int
ai_thread_equal(
    ai_thread_t    *theFirstThread,
    ai_thread_t    *theSecondThread) ;
```

### Description

This function compares one thread identifier to another thread identifier

### Parameters

<i>theFirstThread</i>	First thread identifier
<i>theSecondThread</i>	Second thread identifier.

### Return Values

TRUE (1) if the two threads have the same identifier.

FALSE (0) if the two threads do not have the same identifier.

### Error Conditions

None.

### See Also

None.

## Notes

Does not check whether the corresponding objects currently exist.

## 7.1.21. ai\_thread\_join

### Purpose

Wait for the specified thread to terminate.

### Synopsis

```
int
ai_thread_join(
    ai_thread_t    theThread,    /* IN */
    void            **Status);    /* OUT */
```

### Description

The function causes the calling thread to wait for the termination of a specified thread. A call to this function returns after the specified thread has terminated, either via explicit `ai_thread_exit()`, or via implicit call by returning from the top level (startup) function.

#### Parameters

<i>theThread</i>	Thread to terminate.
<i>Status</i>	Thread exit status.

#### Return Values

The thread's exit status or -1, `errno` is set if a -1 is returned.

#### Error Conditions

None.

#### See Also

None.

#### Notes

None.

### 7.1.22. `ai_thread_self`

#### Purpose

Return calling thread's thread handle.

#### Synopsis

```
ai_thread_t  
ai_thread_self(void);
```

#### Description

Get the handle for the current thread.

#### Parameters

None

#### Return Values

The handle to the callers thread structure.

#### Error Conditions

None.

#### See Also

None.

#### Notes

None.

## 7.1.23. ai\_thread\_sleep

### Purpose

Put a thread to sleep for a specified time.

### Synopsis

```
void
ai_thread_sleep(
    int          Seconds,          /* IN */
    int          Microseconds) ;  /* IN */
```

### Description

This routine causes a thread to delay execution for a specific period of time. This period ends at the current time plus the specified interval. The routine will not return before the end of the period is reached, but may return an arbitrary amount of time after the period has gone by. This is due to system load, thread priorities, and system timer granularity. Specifying an interval of 0 seconds and 0 nanoseconds is allowed and can be used to force the thread to give up the processor or to deliver a pending cancel.

### Parameters

<i>Seconds</i>	Sleep time in seconds.
<i>MicroSeconds</i>	Sleep time in microseconds.

### Return Values

None.

### Error Conditions

None.

### See Also

None.

### Notes

None.

## 7.1.24. ai\_thread\_yield

### Purpose

Yield the processor to another thread.

### Synopsis

```
void
ai_thread_yield(
    void) ;
```

### Description

Notifies the scheduler that the current thread is willing to release its processor to other threads of the same or higher priority. Note that there is no implementation-independent way defined to sleep(), and it's not



guaranteed that the processor will not be immediately given back to this thread.

#### Parameters

*None*

#### Return Values

*None.*

#### Error Conditions

*None.*

#### See Also

*None.*

#### Notes

*None.*

## 7.1.25. ai\_mutex\_destroy

#### Purpose

Destroy a mutex created by ai\_mutex\_init.

#### Synopsis

```
int
ai_mutex_destroy(
    ai_mutex_t    theMutex);    /* IN */
```

#### Description

This function deletes a mutex object that will no longer be referenced. The effect of calling this routine is to reclaim storage allocated when the object was initialized by a previous call to ai\_mutex\_alloc/ai\_mutex\_init. The results of this function are unpredictable if the the mutex object pointed to by theMutex does not exist (i.e., is not currently initialized).

#### Parameters

*theMutex*                      The mutex to be destroyed.

#### Return Values

*None.*

#### Error Conditions

*None.*

#### See Also

*None.*

#### Notes

*None.*

## 7.1.26. ai\_mutex\_init

### Purpose

Initialize a mutex.

### Synopsis

```
int
ai_mutex_init(
    ai_mutex_t      theMutex,      /* OUT */
    ai_mutexattr_t  mutexAttr);    /* IN */
```

### Description

This function creates a mutex and initializes it to the unlocked state. Note that this function and `ai_mutex_alloc` do similar things, except that `ai_mutex_alloc` actually mallocs memory for the mutex variable; this function is called to initialize a structure that already exists for example, a structure of type `ai_mutex_t` that was compiled into the program.

### Parameters

<i>theMutex</i>	The mutex to be initialized.
<i>mutexAttr</i>	Mutex attributes.

### Return Values

Initialized mutex structure.

### Error Conditions

None.

### See Also

None.

### Notes

None.

## 7.1.27. ai\_mutex\_lock

### Purpose

Lock a Mutex.

### Synopsis

```
int
ai_mutex_lock(
    ai_mutex_t      theMutex);    /* IN */
```

### Description

This function locks a mutex. If the mutex is locked when this call is made, the thread blocks waiting for the mutex to become available. The thread that has locked a mutex becomes its current owner and remains the owner until the same thread unlocks it. This routine returns with the mutex in the locked state, and the

current thread as the mutex's owner.

#### Parameters

*theMutex*                      The mutex to be locked.

#### Return Values

0 on success, -1 on error

#### Error Conditions

errno

#### See Also

None.

#### Notes

None.

## 7.1.28. ai\_mutex\_try\_lock

#### Purpose

Non-blocking call to lock a mutex.

#### Synopsis

```
int
ai_mutex_try_lock(
    ai_mutex_t    theMutex);    /* IN */
```

#### Description

This function locks a mutex. If the mutex is locked when this call is made, the calling thread does not wait for the mutex to become available. The thread that has locked a mutex becomes its current owner and remains the owner until the same thread unlocks it. When a thread calls this routine, an attempt is made to immediately lock the mutex. If the mutex is successfully locked, 1 (TRUE) is returned, and the current thread is then the current owner of the mutex. If the mutex is locked by another thread when this function is called, (FALSE) is returned, and the thread does not wait to acquire the lock.

#### Parameters

*theMutex*                      The mutex to be locked.

#### Return Values

0 on success, -1 on error

#### Error Conditions

errno

#### See Also

None.

## Notes

None.

## 7.1.29. ai\_mutex\_unlock

### Purpose

Unlock a mutex.

### Synopsis

```
int
ai_mutex_unlock(
    ai_mutex_t    theMutex);    /* IN */
```

### Description

This function unlocks a mutex, giving other threads a chance to lock it. If no threads are waiting for the mutex, the mutex unlocks with no current owner. If one or more threads are waiting to lock the specified mutex, this function causes one thread to return from its call to ai\_mutex\_lock().

### Parameters

*theMutex*                      The mutex to be unlocked.

### Return Values

0 on success, -1 on error

### Error Conditions

errno

### See Also

None.

## Notes

The results of calling this routine are unpredictable if:

- the specified mutex is actually unlocked at the time of the call
- the specified mutex is currently owned by a thread other than the calling thread

## 7.1.30. ai\_condition\_broadcast

### Purpose

Awaken a thread waiting on a condition variable.

### Synopsis

```
int
ai_condition_broadcast(
    ai_condition_t    theCondition);    /* IN */
```

### Description

This function wakes all threads waiting for a condition variable. Calling this routine implies that data

guarded by the associated mutex has changed. This allows one or more waiting threads to proceed.

#### Parameters

*theCondition*                      Pointer to the condition variable.

#### Return Values

0 on success, -1 on error

#### Error Conditions

errno

#### See Also

None.

#### Notes

None.

### 7.1.31. ai\_condition\_destroy

#### Purpose

Remove a condition variable.

#### Synopsis

```
int
ai_condition_destroy(
    ai_condition_t    theCondition);    /* IN */
```

#### Description

This function deletes a condition object that will no longer be referenced. The effect of calling this routine is to reclaim storage allocated when the object was initialized by a previous call to `ai_condition_init`. The results of this function are unpredictable if the condition object pointed to by `theCondition` does not exist (i.e., was not created and initialized via call to `ai_condition_alloc`).

#### Parameters

*theCondition*                      Pointer to the condition variable.

#### Return Values

0 on success, -1 on error

#### Error Conditions

errno

#### See Also

None.

#### Notes

None.

### 7.1.32. ai\_condition\_init

#### Purpose

Initialize a condition variable.

#### Synopsis

```
int
ai_condition_init(
    ai_condition_t    theCondition,    /* IN */
    ai_condattr_t     conditionAttr);  /* IN */
```

#### Description

This function initializes a pre-existing condition variable. Note that this function and `ai_condition_alloc` do similar things, except that `ai_condition_alloc` actually mallocs memory for the variable; this function is called to initialize a structure that already exists, for example, a structure of type `ai_condition_t` that was compiled into the program or which is contained within another structure.

#### Parameters

<i>theCondition</i>	Pointer to the condition variable.
<i>conditionAttr</i>	Condition attributes.

#### Return Values

0 on success, -1 on error

#### Error Conditions

errno

#### See Also

None.

#### Notes

None.

### 7.1.33. ai\_condition\_signal

#### Purpose

Awaken a single thread waiting on a condition variable.

#### Synopsis

```
int
ai_condition_signal(
    ai_condition_t    theCondition);    /* IN */
```

#### Description

This function wakes one thread waiting for a condition variable. Calling this routine implies that data guarded by the associated mutex has changed, so that it is possible for a single waiting thread to proceed. This function should be called when any thread waiting for the specified condition might find its predicate

true, but only if one thread needs to proceed. Use `ai_condition_broadcast` to wake up all threads that might be waiting on the condition.

#### Parameters

*theCondition*                      Pointer to the condition variable.

#### Return Values

0 on success, -1 on error

#### Error Conditions

`errno`

#### See Also

None.

#### Notes

None.

## 7.1.34. `ai_condition_wait`

#### Purpose

Wait on a condition.

#### Synopsis

```
int
ai_condition_wait(
    ai_condition_t    theCondition,    /* IN */
    ai_mutex_t        theMutex);      /* IN */
```

#### Description

This function unlocks *theMutex*, and suspends the calling thread until the specified condition is likely to be true because some other thread has called `ai_condition_signal` or `ai_condition_broadcast`. It then locks the mutex again, then resumes the thread. There's no guarantee that the condition will be true when the thread is resumed, so it's up to the caller to loop calling `ai_condition_signal` until the condition it's checking is set to the desired state.

#### Parameters

*theCondition*                      Pointer to the condition variable.

*theMutex*                          Pointer to the mutex.

#### Return Values

0 on success, -1 on error

#### Error Conditions

`errno`

#### See Also

**ai\_condition\_signal, ai\_condition\_broadcast**

**Notes**

None.

## 7.1.35. ai\_condition\_timedwait

**Purpose**

Wait on a condition using a timeout

**Synopsis**

```
int
ai_condition_wait(
    ai_condition_t    theCondition,    /* IN */
    ai_mutex_t        theMutex,        /* IN */
    const struct timespec abstime);    /* IN */
```

**Description**

This function unlocks theMutex, and suspends the calling thread until the specified condition is likely to be true because some other thread has called ai\_condition\_signal or ai\_condition\_broadcast or the system time reaches the time specified in <abstime>.

It then locks the mutex again, then resumes the thread.

There's no guarantee that the condition will be true when the thread is resumed, so it's up to the caller to loop calling ai\_condition\_wait or ai\_condition\_timedwait until the condition it's checking is set to the desired state.

**Parameters**

<i>theCondition</i>	Pointer to the condition variable.
<i>theMutex</i>	Pointer to the mutex.
<i>abstime</i>	The time to wait

**Return Values**

0	Success. Returns initialized condition structure
EINVAL, ETIMEOUT	

**Error Conditions**

EINVAL	The value specified by cond, mutex or abstime is invalid
ETIMEOUT	The system time has reached or exceeded the time specified in abstime

**See Also**

None

**Notes**

None.



## 7.2. Data Definitions

This section describes key internal data definitions and all externally used data definitions for the configuration parser. The information provided is:

- Name
- Description
- Format

### 7.2.1. Defines

```
#define HPSS_CFG_MAX_LEVEL          10
#define HPSS_CFG_MAX_CHARS_PER_LINE 512
#define HPSS_CFGOPT_PFTPC           "PFTP Client"
#define HPSS_CFGOPT_PFTPCIF         "PFTP Client Interfaces"
#define HPSS_CFGOPT_MULTINODE       "Multinode Table"
#define HPSS_CFGOPT_NWOPTIONS       "Network Options"
#define HPSS_CFGOPT_PIPEFILE        "Pipe File"
#define HPSS_CFGOPT_LOCALFILE       "Local File Path"
#define HPSS_CFGOPT_COSSELECT       "Select COS"
#define HPSS_CFGOPT_PFTPD           "PFTP Daemon"
#define HPSS_CFGOPT_PSI             "PSI"
#define HPSS_CFGOPT_HSI             "HSI"
#define HPSS_CFGOPT_HTAR            "HTAR"
#define HPSS_CFGOPT_TA              "Transfer Agent"
#define HPSS_DEFAULT_STANZA         "Default"

/*
 * Error values returned from API functions
 */

#define HPSS_CFG_FILEOPEN_ERROR      -7000    /* unable to open specified file */
#define HPSS_CFG_FILE_FMT_ERROR      -7001    /* file does not appear to be a config file */
#define HPSS_CFG_FILE_MEM_ERROR      -7002    /* memory allocation problem */
#define HPSS_CFG_FILE_PARSE_ERR01    -7003    /* parse error - too many levels */
#define HPSS_CFG_FILE_PARSE_ERR02    -7004    /* syntax error: "{" not preceded by "=" */
#define HPSS_CFG_FILE_PARSE_ERR03    -7005    /* parse error - extraneous closing "}" */
#define HPSS_CFG_FILE_PARSE_ERR04    7006    /* parse error - missing closing "}" */
```

```

#define HPSS_CFG_FILEOPEN_ERROR      -7000    /* unable to open specified file */
#define HPSS_CFG_FILE_NO_KEY         -7007    /* No Key Found */
#define HPSS_CFG_FILE_NO_VALUE       -7008    /* No Value Found */
#define HPSS_CFG_FILE_NOT_CMPD       -7009    /* Not Compound Stanza */
#define HPSS_CFG_FILE_SECTION_EMPTY -7010    /* Empty Compound Stanza */

#define SEARCH_BY_KEY                 0
#define SEARCH_BY_VALUE               1
#define SEARCH_THIS_LEVEL             0
#define SEARCH_SUB_LEVELS             1
#define CASE_SENSITIVE                0
#define CASE_INSENSITIVE              1

```

## 7.2.2. hpss\_cfg\_stanza\_t

### Description

**hpss\_cfg\_stanza\_t** is a linked-list of structures used to store data extracted from a specific stanza in the configuration file. Refer to **HPSS.conf(7)** for definitions and terminology.

### Format

**hpss\_cfg\_stanza\_t** has the following format:

```

typedef struct _hpss_cfg_stanza_t hpss_cfg_stanza_t;

struct _hpss_cfg_stanza_t {
    hpss_cfg_stanza_t  *next;
    int                level;
    int                type;
    unsigned32         flags32;
    unsigned32         lineNumber;
    u_signed64         val64;
    char               *keyString;
    char               *keyValue;
    int                stanzaCount;
    hpss_cfg_stanza_t *stanzaList;
};

```

#### next

Linked list of substanzas for this level .

#### level

Level of this entry.

#### type

Enumerated type for this entry

**Flags32**

Application-specific flags reserved for application(s)

**LineNum**

Line number within file where stanza starts

**Val64**

Application-specific info. Reserved for application(s)

**KeyString**

(malloc-ed) flag string for this entry

**KeyValue**

(malloc-ed) NULL or value string for this entry

**substanzaCount**

Number of substanzas for this entry

**substanzaList**

Head of linked list of substanzas for this entry

**Format**

**addr\_array\_t** and **addr\_array\_p** have the following format:

```
typedef struct addr_array_desc {
    char          IFName[20] ;
    char          DotAddress[16] ;
    unsigned long  Flags;
} addr_array_t, *addr_array_p;
```

**IFName**

The name of the interface (*e.g.*, en0).

**DotAddress**

The dotted IP address of the interface.

**Flags**

Flags associated with the address (See **ifreq.ifr\_flags** in **net/if.h**).



## Chapter 8. Real Time Monitoring

This chapter specifies the Real Time Monitoring library calls and the programming interface.

Specifically, the following information is provided:

Application Programming Interface (API) for the Real Time Monitoring Service

- Name
- Syntax
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Clients
- Notes

RTM Library Routines

- Name
- Syntax
- Description
- Parameters
- Return Values
- Error Conditions
- See Also
- Clients
- Notes
- Data Definitions

### 8.1. RTM API Interfaces

#### 8.1.1. `rtm_GetRequestEntries`

**Syntax**

```
[idempotent, nontransactional]
```

```

signed32 rtm_GetRequestEntries(
    hpss_connect_handle_t      CnhPtr,          /* IN */
    hpss_reqid_t               ReqId,           /* IN */
    unsigned32                  ListName,        /* IN */
    unsigned32                  Flags,           /* IN */
    hpss_reqid_t               RReqId,          /* IN */
    rtm_object_t                *ObjectPtr,      /* IN */
    rtm_req_array_t            **ReqArrayPtr    /* OUT */
);

```

## Description

This API is the interface for returning the internally-stored request information from an HPSS Server.

## Parameters

<i>CnhPtr</i>	The connection handle.
<i>ReqId</i>	This requests request id.
<i>ListName</i>	This identifies the specific request list of interest.
<i>Flags</i>	An integer that determines what information to return. Only one flag value may be set.  RTM_FLAG_ONE_REQUEST = 0x1; Returns info on the request id in RReqId.  RTM_FLAG_ALL_REQUESTS = 0x2; Returns info on all requests known to the HPSS Server.  RTM_FLAG_OBJECT = 0x4; Returns info on all requests relating to ObjectPtr.
<i>RReqId</i>	The requested request id.
<i>ObjectPtr</i>	A union of objects about which request information is to be returned. Initially this will only be used for the bitfile id and the pv name.
<i>ReqArrayPtr</i>	The returned request information - a conformant array.

## Return Values

Zero indicates success. A value less than zero indicates an error and is a code that defines the error.

## Error Conditions

HPSS_EBADCONN	Problem with server connection.
HPSS_EINVAL	Invalid argument to API.
HPSS_EMUTEXPROBLEM	Mutex error in locking or unlocking the request list.
HPSS_ENOERROR	Successful return.
HPSS_ENOMEM	hpss_RPCMalloc failed to allocate memory.
HPSS_ENOTREADY	RTM list has not been initialized for the given server.
HPSS_EPERM	Unauthorized caller.
HPSS_ESYSTEM	Software error.

## See Also

None.

## Clients

The RTM utility called `rtmu`.

## Notes

None.

## 8.2. RTM API Data Definitions

This section describes key internal data definitions and all externally used data definitions which are provided by this subsystem. A data definition may be represented by constructs such as data structures and constants. For each data definition, a description, format (including parameter descriptions), and clients which access the data definition are provided.

The information provided is:

- Name
- Description
- Format
- Clients

### 8.2.1. RTM Object Type Enumeration - `rtm_objectname_t`

#### Description

`rtm_objectname_t` defines an enumerated set of object types which may appear in the `rtm_object_t` structure.

#### Format

```
typedef enum {  
    RTM_OBJECT_INVALID=0,  
    RTM_OBJECT_BFID=1,  
    RTM_OBJECT_PVNAME=2  
} rtm_objectname_t;
```

#### Clients

The RTM utility called `rtmu`.

### 8.2.2. RTM Object Type - `rtm_object_t`

#### Description

This is the definition of the union `rtm_object_t` that holds the particular object type about which information is being requested.

#### Format

```
typedef union switch(rtm_objectname_t ObjectName) objects_u {
```

```

        case RTM_OBJECT_INVALID:
            unsigned32      one;
        case RTM_OBJECT_BFID:
            hpssoid_t       BfId;
        case RTM_OBJECT_PVNAME:
            char             PVName[HPSS_PV_NAME_SIZE];
    } rtm_object_t;

```

**one**

Just to have a completely separate case for invalid.

**BfId**

The bitfile id about which information is being requested.

**PVName**

The Physical Volume Name about which information is being requested.

**Clients**

The RTM utility called rtmu.

## 8.2.3. Request Array - rtm\_req\_array\_t

**Description**

This is the definition of the conformant array that is returned in the RTM API. This array contains the server request information.

**Format**

```

typedef struct req_array {
    struct {
        u_int                      ReqArray_len;
        rtm_req_array_entry_t     *ReqArray_val;
    } ReqArray
} rtm_req_array_t;

```

**Size**

The size of the conformant array.

**ReqArray**

The conformant array of request entry data.

**Clients**

The RTM utility called rtmu.

## 8.2.4. Request Entry - rtm\_req\_array\_entry\_t

**Description**

The **rtm\_req\_array\_entry\_t** defines one request entry in the returned conformant array of request entries. There may be one or more of these entries per request from a server, depending on the number of wait states



that this request has outstanding. As an example, the core server could have eight such entries to describe eight waits for mounts on a write to an 8-way tape class of service.

#### Format

```
typedef struct reqentry {
    hpss_reqid_t      ReqId;
    rtm_request_code_t ReqCode;
    rtm_state_t       ReqState;
    char              ServerDescName[HPSS_MAX_DESC_NAME];
    uuid_t            ServerId;
    timestamp_sec_t   StartTime;
    timestamp_sec_t   UpdateTime;
    rtm_serverspecific_t ServerSpecific;
    unsigned32        ThreadId;
    unsigned32        ProcessId;
    rtm_wait_reason_t WaitReason;
    rtm_wait_resource_t WaitResource;
} rtm_req_array_entry_t;
```

#### ReqId

Requested request id.

#### ReqCode

The server-specific request code of the returned request. See the `rtm_request_code_t` definition below.

#### ReqState

The state of the request in the server. See the `rtm_state_t` definition below.

#### ServerDescName

The descriptive name of the executing server e.g. CORE.

#### ServerId

The server id of the executing server.

#### StartTime

The time this request started in this server.

#### UpdateTime

The time this request information was last updated by this server.

#### ServerSpecific

A server-specific structure that has information about the request that is appropriate to that server.

#### ThreadID

The server thread id that is handling the specific request wait-state being described.

#### ProcessId

The server process id that is handling the specific request wait-state being described. The HPSS Mover has processes and not threads.

### WaitReason

Server-specific integer indicating reason for waiting. See the definition of `rtm_wait_reason_t` below.

### WaitResource

This field is the resource being waited on. See the definition of `rtm_wait_resource_t` below.

### Clients

The RTM utility called `rtmu`.

## 8.2.5. Request Code Enumeration - `rtm_request_code_t`

### Description

The `rtm_request_code_t` defines an enumerated set of request codes. Each server has its own set of request codes with its own numbering range.

### Format

```
typedef enum {
    RTM_INVALID_REQ = 0,
    RTM_CORE_CREATE_REQ = 1,
    RTM_CORE_OPEN_REQ = 2,
    RTM_CORE_CLOSE_REQ = 3,
    RTM_CORE_UNLINK_REQ = 4,
    RTM_CORE_READ_REQ = 5,
    RTM_CORE_WRITE_REQ = 6,
    RTM_CORE_SET_ATTRIB_BF_REQ = 7,
    RTM_CORE_SET_ATTRIB_BF_O_REQ = 8,
    RTM_CORE_GET_ATTRIB_BF_REQ = 9,
    RTM_CORE_GET_ATTRIB_BF_O_REQ = 10,
    RTM_CORE_MIGRATE_REQ = 11,
    RTM_CORE_STAGE_REQ = 12,
    RTM_CORE_STAGE_BACKG_REQ = 13,
    RTM_CORE_CLEAR_REQ = 14,
    RTM_CORE_PURGE_REQ = 15,
    RTM_CORE_MODIFY_REQ = 16,
    RTM_CORE_COPYFILE_REQ = 17,
    RTM_CORE_TAPE_WRITE_REQ = 18,
    RTM_CORE_TAPE_READ_REQ = 19,
    RTM_CORE_TAPE_MOVE_SEG_REQ = 20,
    RTM_CORE_TAPE_POSITION_REQ = 21,
    RTM_CORE_TAPE_CREATE_RSRC_REQ = 22,
    RTM_CORE_TAPE_DELETE_RSRC_REQ = 23,
    RTM_CORE_TAPE_WTM = 24,
    RTM_CORE_TAPE_SET_ATTRS_REQ = 25,
    RTM_CORE_TAPE_GET_ATTRS_REQ = 26,
    RTM_CORE_TAPE_MOUNT_REQ = 27,
    RTM_CORE_TAPE_DISMOUNT_REQ = 28,
    RTM_CORE_TAPE_STARTMOUNT_REQ = 29,
    RTM_CORE_DISK_WRITE_REQ = 30,
    RTM_CORE_DISK_READ_REQ = 31,
    RTM_CORE_DISK_MOVE_SEG_REQ = 32,
    RTM_CORE_DISK_CREATE_RSRC_REQ = 33,
```

```

RTM_CORE_DISK_DELETE_RSRC_REQ = 34,
RTM_CORE_DISK_SET_ATTRS_REQ = 35,
RTM_CORE_DISK_GET_ATTRS_REQ = 36,
RTM_CORE_DISK_MOUNT_REQ = 37,
RTM_CORE_DISK_DISMOUNT_REQ = 38,
RTM_CORE_PVL_VERIFY_REQ = 39,
RTM_CORE_SHUTDOWN_REQ = 40,
RTM_CORE_PVL_DISMNT_HANDLER_REQ = 41,
RTM_CORE_OPEN_CONNECTION_REQ = 42,
RTM_CORE_CLOSE_CONNECTION_REQ = 43,
RTM_CORE_BEGIN_SESSION_REQ = 44,
RTM_CORE_END_SESSION_REQ = 45,
RTM_CORE_SSM_CONNECT_REQ = 46,
RTM_CORE_SSM_NOTIFICATION_REQ = 47,
RTM_CORE_SIGNAL_HANDLER_REQ = 48,
RTM_CORE_SERVER_GET_ATTR_REQ = 49,
RTM_CORE_SERVER_SET_ATTR_REQ = 50,
RTM_CORE_SERVER_INIT_REQ = 51,
RTM_CORE_GET_SCSTATS_REQ = 52,
RTM_CORE_RESET_MAPS_REQ = 53,
RTM_CORE_CACHE_FLUSH_REQ = 54,
RTM_CORE_PVL_CALLBACK_REQ = 55,
RTM_CORE_PING_SSM_REQ = 56,
RTM_CORE_GET_VV_COND_REQ = 57,
RTM_CORE_SET_VV_COND_REQ = 58,
RTM_CORE_GET_PV_SOID_REQ = 59,
RTM_CORE_RESERVED_1 = 60,
RTM_CORE_RESERVED_2 = 61,
RTM_MVR_READ_REQ = 62,
RTM_MVR_WRITE_REQ = 63,
RTM_MVR_DEV_GETATTR_REQ = 64,
RTM_MVR_DEV_SETATTR_REQ = 65,
RTM_MVR_UNLOAD_REQ = 66,
RTM_MVR_FLUSH_REQ = 67,
RTM_MVR_WRITETM_REQ = 68,
RTM_MVR_READLABEL_REQ = 69,
RTM_MVR_WRITELABEL_REQ = 70,
RTM_MVR_CLEAR_REQ = 71,
RTM_MVR_LOAD_REQ = 72,
RTM_MVR_LOADDISPLAY_REQ = 73,
RTM_GK_OPEN_REQ = 74,
RTM_GK_CLOSE_REQ = 75,
RTM_GK_CREATE_REQ = 76,
RTM_GK_CREATE_COMPLETE_REQ = 77,
RTM_GK_STAGE_REQ = 78,
RTM_GK_STAGE_COMPLETE_REQ = 79,
RTM_GK_CLEAN_UP_CLIENT_REQ = 80,
RTM_GK_MONITOR_REQ = 81,
RTM_GK_PASS_THRU_REQ = 82,
RTM_GK_QUERY_REQ = 83,
RTM_GK_READ_POLICY_REQ = 84,
RTM_GK_CLOSE_CONNECTION_REQ = 85,
RTM_GK_TERM_CONNECT_THREAD_REQ = 86,

```

```

    RTM_GK_OPEN_CONNECTION_REQ = 87,
    RTM_GK_GET_ATTR_REQ = 88,
    RTM_GK_SET_ATTR_REQ = 89,
    RTM_GK_SRVR_GET_ATTR_REQ = 90,
    RTM_GK_SRVR_SET_ATTR_REQ = 91,
    RTM_GK_SSM_CONNECT_REQ = 92,
    RTM_GK_INITIALIZE_REQ = 93,
    RTM_GK_SIGNALTHREAD_REQ = 94,
    RTM_GK_SHUTDOWN_REQ = 95,
    RTM_GK_SSM_NOTIFICATION_REQ = 96
} rtm_request_code_t;

```

#### Clients

The RTM utility called rtmu.

### 8.2.6. Request State - rtm\_state\_t

#### Description

The `rtm_state_t` defines the enumerated states that the request may be in.

#### Format

```

typedef enum {
    RTM_REQ_INVALID = 0,
    RTM_REQ_INITIALIZING = 1,
    RTM_REQ_QUEUED = 2,
    RTM_REQ_IN_PROGRESS = 3,
    RTM_REQ_BLOCKED = 4,
    RTM_REQ_SUSPENDED = 5,
    RTM_REQ_ERROR_STATE = 6,
    RTM_REQ_COMPLETED = 7,
    RTM_REQ_NOT_INUSE = 8
} rtm_state_t;

```

#### Clients

The RTM utility called rtmu.

### 8.2.7. Server Request Information - rtm\_serverspecific\_t

#### Description

This union has a different request-information structure for each participating HPSS server. In this way each server is returning different kinds of information about a request.

#### Format

```

typedef union switch(rtm_server_type_t ServerType) Attribs_u {
    case RTM_SERVER_INVALID:
        char RTMComment[RTM_COMMENT_LEN];
    case RTM_CORE_SERVER:
        rtm_core_req_attr_t COREReqInfo;
    case RTM_MVR_SERVER:
        rtm_mvr_req_attr_t MVRReqInfo;
}

```

```

        case RTM_GK_SERVER:
            rtm_gk_req_attr_t                GKReqInfo;
    } rtm_serverspecific_t;

```

### RTMComment

rtm\_GetRequestEntries returns an error message here if an invalid server is specified as ServerType.

### COREReqInfo

This is the CORE-specific structure that returns request information from the Core Server.

### MVRReqInfo

This is the MVR-specific structure that returns request information from the Mover.

### GKReqInfo

This is the GK-specific structure that returns request information from the GateKeeper.

## Clients

The RTM Utility called rtmu.

## 8.2.8. Wait Reason Enumeration - rtm\_wait\_reason\_t

### Description

**rtm\_wait\_reason\_t** defines the enumerated set of server-specific wait reasons that HPSS servers have for waiting on a resource.

### Format

```

typedef enum {
    RTM_WAIT_REASON_INVALID = 0,
    RTM_WAIT_CORE_SS_CREATE = 1,
    RTM_WAIT_CORE_SS_DELETE = 2,
    RTM_WAIT_CORE_SS_DELETELIST = 3,
    RTM_WAIT_CORE_SS_READ = 4,
    RTM_WAIT_CORE_SS_WRITE = 5,
    RTM_WAIT_CORE_SS_GETATTRS = 6,
    RTM_WAIT_CORE_SS_SETATTRS = 7,
    RTM_WAIT_CORE_SS_SCLASS_STATS = 8,
    RTM_WAIT_CORE_SS_BEGINSESS = 9,
    RTM_WAIT_CORE_SS_ENDSESS = 10,
    RTM_WAIT_CORE_SS_STARTMNT = 11,
    RTM_WAIT_CORE_NS_GETFSATTRS = 12,
    RTM_WAIT_CORE_NS_GETNAME = 13,
    RTM_WAIT_CORE_NS_SETATTRS = 14,
    RTM_WAIT_CORE_FOR_FILE = 15,
    RTM_WAIT_CORE_MOVER_REPLY = 16,
    RTM_WAIT_CORE_ASYNC_SCHDL = 17,
    RTM_WAIT_CORE_PVL_MOUNT = 18,
    RTM_WAIT_CORE_PVL_RESERVED_3 = 19,
    RTM_WAIT_CORE_PVL_GETATTRS = 20,
    RTM_WAIT_CORE_PVL_ALLOCATE = 21,
    RTM_WAIT_CORE_PVL_DEALLOCATE = 22,

```

```

RTM_WAIT_CORE_PVL_CONNECT = 23,
RTM_WAIT_CORE_PVL_CLOSE_CONNECT = 24,
RTM_WAIT_CORE_PVL_RESERVED_2 = 25,
RTM_WAIT_CORE_PVL_CANCEL_JOBS = 26,
RTM_WAIT_CORE_PVL_DISMOUNT = 27,
RTM_WAIT_CORE_PVL_MOUNT_NEW = 28,
RTM_WAIT_CORE_PVL_MOUNT_ADD = 29,
RTM_WAIT_CORE_PVL_MOUNT_COMMIT = 30,
RTM_WAIT_CORE_PVL_MNT_COMPLETE = 31,
RTM_WAIT_CORE_PVL_Q_GET_ATTRS = 32,
RTM_WAIT_CORE_PVL_VOL_GET_ATTRS = 33,
RTM_WAIT_CORE_PVL_RESERVED_1 = 34,
RTM_WAIT_CORE_MVR_READ_DATA = 35,
RTM_WAIT_CORE_MVR_WRITE_DATA = 36,
RTM_WAIT_CORE_MVR_WRITE_TM = 37,
RTM_WAIT_CORE_MVR_POSITION = 38,
RTM_WAIT_CORE_MVR_ASSIGN_IP = 39,
RTM_WAIT_CORE_MVR_RESERVED_1 = 40,
RTM_WAIT_CORE_SESSION_RESV_1 = 41,
RTM_WAIT_CORE_SESSION_BUSY = 42,
RTM_WAIT_CORE_SESSION_ASSIGN_VV = 43,
RTM_WAIT_CORE_SSM_CONNECT = 44,
RTM_WAIT_CORE_SSM_NOTIFICATION = 45,
RTM_WAIT_CORE_SEG_CACHE_ENTRY = 46,
RTM_WAIT_CORE_VV_CACHE_ENTRY = 47,
RTM_WAIT_CORE_VV_GETATTRS = 48,
RTM_WAIT_CORE_PV_CACHE_ENTRY = 49,
RTM_WAIT_CORE_MAP_CACHE_ENTRY = 50,
RTM_WAIT_CORE_PV_HANDOFF = 51,
RTM_WAIT_CORE_RESERVED_5 = 52,
RTM_WAIT_CORE_UNREG_SERVICE = 53,
RTM_WAIT_CORE_RESERVED_4 = 54,
RTM_WAIT_CORE_PV_REFERENCE = 55,
RTM_WAIT_CORE_PV_DISMOUNT = 56,
RTM_WAIT_CORE_SEG_FREE = 57,
RTM_WAIT_CORE_RESERVED_2 = 58,
RTM_WAIT_CORE_RESERVED_3 = 59,
RTM_WAIT_MVR_DEVICE_AVAIL = 60,
RTM_WAIT_MVR_DEVICE_OPEN = 61,
RTM_WAIT_MVR_DEVICE_IO = 62,
RTM_WAIT_MVR_DEVICE_POS = 63,
RTM_WAIT_MVR_DEVICE_GET_ABSPOS = 64,
RTM_WAIT_MVR_DEVICE_SYNC = 65,
RTM_WAIT_MVR_DEVICE_WTM = 66,
RTM_WAIT_MVR_DEVICE_CLOSE = 67,
RTM_WAIT_MVR_DEVICE_REWIND = 68,
RTM_WAIT_MVR_DEVICE_UNLOAD = 69,
RTM_WAIT_MVR_DEVICE_DISPLAY = 70,
RTM_WAIT_MVR_CLIENT_CONN = 71,
RTM_WAIT_MVR_CLIENT_SEND = 72,
RTM_WAIT_MVR_CLIENT_RECV = 73,
RTM_WAIT_MVR_CLIENT_CLOSE = 74,
RTM_WAIT_MVR_PROT_RECV = 75,

```

```

RTM_WAIT_MVR_PROT_SEND = 76,
RTM_WAIT_MVR_ACTIVE_INFO = 77,
RTM_WAIT_MVR_IOR_SEND = 78,
RTM_WAIT_MVR_IOD_RECV = 79,
RTM_WAIT_GK_SITE_OPEN = 80,
RTM_WAIT_GK_SITE_OPEN_STATS = 81,
RTM_WAIT_GK_SITE_CLOSE = 82,
RTM_WAIT_GK_SITE_CREATE = 83,
RTM_WAIT_GK_SITE_CREATE_STATS = 84,
RTM_WAIT_GK_SITE_CREATE_COMPLET = 85,
RTM_WAIT_GK_SITE_STAGE = 86,
RTM_WAIT_GK_SITE_STAGE_STATS = 87,
RTM_WAIT_GK_SITE_STAGE_COMPLETE = 88,
RTM_WAIT_GK_SITE_MONITOR = 89,
RTM_WAIT_GK_SITE_PASS_THRU = 90,
RTM_WAIT_GK_SITE_READ_POLICY = 91,
RTM_WAIT_GK_SITE_SHUTDOWN = 92,
RTM_WAIT_GK_RETRY = 93,
RTM_WAIT_GK_ASYNC_QUEUE = 94,
RTM_WAIT_GK_MO_LOCK = 95,
RTM_WAIT_GK_SRV_STATE_LOCK = 96,
RTM_WAIT_GK_SSM_STATE_LOCK = 97,
RTM_WAIT_GK_SSM_CONNECT = 98,
RTM_WAIT_GK_SSM_NOTIFICATION = 99,
RTM_WAIT_GK_SIGNAL = 100,
RTM_WAIT_GK_SHUTDOWN = 101,
RTM_WAIT_GK_SHUTDOWN_GK_IF = 102,
RTM_WAIT_GK_SHUTDOWN_ADMIN_IF = 103,
RTM_WAIT_GK_SHUTDOWN_AVAL_IF = 104
} rtm_wait_reason_t;

```

## Clients

The RTM Utility called rtmu.

## 8.2.9. Wait Resource - rtm\_wait\_resource\_t

### Description

This union holds the variety of possible objects that an HPSS server might be waiting on.

### Format

```

typedef union switch(rtm_wait_type_t WaitType) WaitResource_u {
    case RTM_WAIT_INVALID:
        char                                RTMComment[RTM_COMMENT_LEN];
    case RTM_WAIT_NONE:
        char                                NOcomment[HPSS_MAX_COMMENT_LENGTH];
    case RTM_WAIT_COMMENT:
        char                                Comment[HPSS_MAX_COMMENT_LENGTH];
    case RTM_WAIT_SERVER:
        uuid_t                              ServerId;
    case RTM_WAIT_SOID:
        hpssoid_t                           Soid;
}

```

```

    case RTM_WAIT_NAME:
        char Name[RTM_NAME_LEN] ;
    case RTM_WAIT_JOB:
        unsigned32 PvlJob;
    case RTM_WAIT_ID:
        unsigned32 ID;
    case RTM_WAIT_ADDRESS:
        address_t Address;
} rtm_wait_resource_t;

```

#### **RTMComment**

Comment space for rtm\_GetRequestEntries to leave error message.

#### **NOcomment**

Comment space for server storing wait state if not really 'waiting' on an event.

#### **Comment**

Comment space for server storing wait state when waiting on an event.

#### **ServerId**

This is the uuid of the HPSS server being waited-on. For example, the CORE might be waiting on the MVR, so the uuid of the MVR would be stored here.

#### **Soid**

This could be an object like a virtual volume.

#### **Name**

Any name, such as a physical volume name.

#### **PvlJob**

PVL Job Id.

#### **ID**

Any id, such as a volume id.

#### **Address**

Client device address.

### **Clients**

The RTM utility called rtmu.

## **8.2.10. Wait Type Enumeration - rtm\_wait\_type\_t**

### **Description**

This is an enumeration of the possible objects that an HPSS server might be waiting on.

### **Format**

```
typedef enum {
```



```

    RTM_WAIT_INVALID = 0,
    RTM_WAIT_NONE = 1,
    RTM_WAIT_COMMENT = 2,
    RTM_WAIT_SERVER = 3,
    RTM_WAIT_SOID = 4,
    RTM_WAIT_NAME = 5,
    RTM_WAIT_JOB = 6,
    RTM_WAIT_ID = 7,
    RTM_WAIT_ADDRESS = 8,
    RTM_WAIT_MAXVALUE = 9
} rtm_wait_type_t;

```

## Clients

The RTM utility called rtmu.

## 8.2.11. CORE Request - rtm\_core\_req\_attr\_t

### Description

**rtm\_core\_req\_attr\_t** is the core-specific structure returned by the Core Server.

### Format

```

typedef struct rtm_core_req_attr {
    unsigned32    CoreServer_bitlist;
    unsigned32    Flags;
    unsigned32    SubsystemId;
    unsigned32    UserRealmId;
    u_signed64    FilesetId;
    char          FilesetName[RTM_FILESET_NAME_LENGTH];
    hpssoid_t     BfId;
    char          BfPathname[RTM_BF_PATHNAME_LENGTH];
    unsigned32    UserId;
    unsigned32    UserGId;
    unsigned32    COSId;
    unsigned32    BFS_SClassId;
    u_signed64    BFS_BytesRequested;
    u_signed64    BFS_BytesMoved;

    signed32      PVLJobId;
    uuid_t        MvrId;
    signed32      DeviceId;
    hpssoid_t     Segment;
    hpssoid_t     VV;
    char          PVName[HPSS_PV_NAME_SIZE];
    relative_address_t CurrentRelPosition;
    absolute_address_t CurrentAbsPosition;
    signed32      ExcessVVsInUse;
    unsigned32    FamilyId;
    unsigned32    SS_SClassId;
    u_signed64    SS_BytesRequested;
    u_signed64    SS_BytesMoved;
} rtm_core_req_attr_t;

```

### **CoreServer\_bitlist**

This is a bitlist indicating which fields within `rtm_core_req_attr_t` are valid. The bit position definitions are as follows:

```
RTM_CORE_SUBSYSTEMID_BIT = 0;
RTM_CORE_REALMID_BIT = 1;
RTM_CORE_FILESETID_BIT = 2;
RTM_CORE_FILESETNAME_BIT = 3;
RTM_CORE_BFID_BIT = 4;
RTM_CORE_BFPATHNAME_BIT = 5;
RTM_CORE_USERID_BIT = 6;
RTM_CORE_USERGID_BIT = 7;
RTM_CORE_COSID_BIT = 8;
RTM_CORE_BFS_SCLASSID_BIT = 9;
RTM_CORE_BFS_BYTESREQUESTED_BIT = 10;
RTM_CORE_BFS_BYTESMOVED_BIT = 11;
RTM_CORE_PVLJOBID_BIT = 12;
RTM_CORE_MVRID_BIT = 13;
RTM_CORE_DEVICEID_BIT = 14;
RTM_CORE_SEGMENT_BIT = 15;
RTM_CORE_VV_BIT = 16;
RTM_CORE_PVNAME_BIT = 17;
RTM_CORE_CURRENTRELPOS_BIT = 18;
RTM_CORE_CURRENTABSPOS_BIT = 19;
RTM_CORE_EXCESSVVS_BIT = 20;
RTM_CORE_FAMILYID_BIT = 21;
RTM_CORE_SS_SCLASSID_BIT = 22;
RTM_CORE_SS_BYTESREQUESTED_BIT = 23;
RTM_CORE_SS_BYTESMOVED_BIT = 24;
```

### **Flags**

For future use.

### **SubsystemId**

The Subsystem Id.

**UserRealmId**

The realm id of the file involved in this request.

**FilesetId**

The fileset id of the file involved in this request.

**FilesetName**

The fileset pathname (if available) of the file involved in this request.

**BfId**

The Bitfile Id of the file involved in this request.

**BfPathname**

The file pathname (if available) of the file relative to the fileset.

**UserId**

The Unix userid (if available) of the user involved in this request.

**UserGid**

The Unix groupid (if available) of the user involved in this request.

**COSId**

The Class of Service of the file involved in this request.

**BFS\_SClassId**

The Storage Class id of the file involved in this request.

**BFS\_BytesRequested**

The number of bytes requested by this request.

**BFS\_BytesMoved**

The number of bytes already moved by this request.

**PVLJobId**

The PVL Job Id associated with this request.

**MvrId**

The Mover Id associated with the Core Server wait state for this request.

**DeviceId**

The ID of the device associated with the Core Server wait state for this request.

**Segment**

The Storage Segment associated with the Core Server wait state for this request.

**VV**

The Virtual Volume associated with the Core Server wait state for this request.

### **PVName**

The Physical Volume name associated with the Core Server wait state for this request.

### **CurrentRelPosition**

The Current Relative Position.

### **CurrentAbsPosition**

The Current Absolute Position.

### **ExcessVVsInUse**

This wait state occurs when the number of tape VVs in use exceeds the quota for the storage class. Additional requests to assign a tape VV to a session block until the quota is restored.

### **FamilyId**

The File Family Id.

### **SS\_SClassId**

The Storage Class Id.

### **SS\_BytesRequested**

The number of data bytes requested to be moved by this request.

### **SS\_BytesMoved**

The number of data bytes already moved by this request.

## **Clients**

The RTM utility called rtmu.

## **8.2.12. Mover Request - rtm\_mvr\_req\_attr\_t**

### **Description**

This is the mover-specific structure that returns information from the Mover about this request.

### **Format**

```
typedef struct rtm_mvr_req_attr {
    unsigned32  Mvr_bitlist;
    unsigned32  Flags;
    u_signed64  InitialTransferOffset;
    u_signed64  CurrentTransferOffset;
    long        DeviceId;
    char        VolumeId[PV_NAME_LENGTH];
    u_signed64  BytesRequested;
    u_signed64  BytesMoved;
} rtm_mvr_req_attr_t;
```

### **Mvr\_bitlist**

This is a bitlist indicating which fields within **rtm\_mvr\_req\_attr\_t** are valid. The bit position definitions are as follows:

```

RTM_MVR_INITIALXFEROFFSET_BIT = 0;
RTM_MVR_CURRENTXFEROFFSET_BIT = 1;
RTM_MVR_DEVICEID_BIT = 2;
RTM_MVR_VOLUMEID_BIT = 3;
RTM_MVR_BYTESREQUESTED_BIT = 4;
RTM_MVR_BYTESMOVED = 5;

```

### **Flags**

For future use.

### **InitialTransferOffset**

The initial transfer offset of the request.

### **CurrentTransferOffset**

The current transfer offset of the request.

### **DeviceId**

The device id of the associated device.

### **VolumeId**

The volume id of the associated volume.

### **BytesRequested**

The number of bytes requested by this request.

### **BytesMoved**

The number of bytes moved thus far by this request.

## **Clients**

The RTM utility called rtmu.

## **8.2.13. Gatekeeper Request - rtm\_gk\_req\_attr\_t**

### **Description**

This is the gk-specific structure that returns information from the GateKeeper about this request.

### **Format**

```

typedef struct rtm_gk_req_attr {
    unsigned32          Gk_bitlist;
    unsigned32          Flags;
    unsigned32          AuthorizedCaller;
    hpssoid_t           BitFileId;
    hpss_uid_t          ConnectionId;
    hpss_uid_t          ClientConnectId;
    hpss_uid_t          ControlNo;
    unsigned32          RealmId;
}

```

```

        unsigned32      GroupId;
        hpss_sec_sockaddr_t  HostAddr;
        unsigned32      OFlag;
        unsigned32      RequestType;
        unsigned32      StageFlags;
        u_signed64      StageLength;
        u_signed64      StageOffset;
        unsigned32      StageStorageLevel;
        unsigned32      UserId;
        unsigned32      WaitTime;
} rtm_gk_req_attr_t;

```

### **Gk\_bitlist**

This is a bitlist indicating which fields within **rtm\_gk\_req\_attr\_t** are valid. The bit position definitions are as follows:

```

RTM_GK_AUTHORIZEDCALLER_BIT = 0;
RTM_GK_BITFILEID_BIT = 1;
RTM_GK_CLIENTCONNECTID_BIT = 2;
RTM_GK_CONNECTIONID_BIT = 3;
RTM_GK_CONTROLNO_BIT = 4;
RTM_GK_REALMID_BIT = 5;
RTM_GK_GROUPID_BIT = 6;
RTM_GK_HOSTADDR_BIT = 7;
RTM_GK_OFLAG_BIT = 8;
RTM_GK_REQUESTTYPE_BIT = 9;
RTM_GK_STAGEFLAGS_BIT = 10;
RTM_GK_STAGELENGTH_BIT = 11;
RTM_GK_STAGEOFFSET_BIT = 12;
RTM_GK_STAGESTORAGELEVEL_BIT = 13;
RTM_GK_USERID_BIT = 14;
RTM_GK_WAITTIME_BIT = 15;

```

### **Flags**

For future use.

### **AuthorizedCaller**

The authorized caller for this call.

### **BitFileId**

The unique bitfile identifier associated with a file. This field is zero for file creates.

### **ConnectionId**

The unique connection identifier associated with the Gatekeeper's client process (usually CORE).

### **ClientConnectId**

The unique connection identifier associated with the end user client process, namely the connection Id between the end client and CORE.

### **ControlNo**

The unique gatekeeper request identifier. Used as a unique key and for identifying a particular client request.

### **RealmId**

The HPSS-defined integer identifier of the Realm. This should be used in conjunction with the UserId to uniquely identify a user.

### **GroupId**

The UNIX Group Id of the user opening the file.

### **HostAddr**

The address of the host that the user is on.

### **Oflag**

Specifies the open file status flags (e.g. O\_RDONLY).

### **RequestType**

The type of gatekeeping request. Possible request types are:

RTM\_GK\_CREATE\_REQUEST

RTM\_GK\_OPEN\_REQUEST

RTM\_GK\_STAGE\_REQUEST

### **StageFlags**

Controls the behavior of the stage request. Valid values include:

CORE\_STAGE\_ALL: Stage entire file.

CORE\_ASYNC\_CALL: Return after initiating stage.

### **StageLength**

Specifies the length of the data to be staged.

### **StageOffset**

Specifies the offset of the start of the data to be staged.

### **StageStorageLevel**

Identifies the level in the storage hierarchy to which the data is to be staged.

### **UserId**

The UNIX UserId of the user that is opening the file. This should be used in conjunction with the RealmId to uniquely identify a user.

### **WaitTime**

The number of seconds the Client API has to wait before retrying an open request that returned the status HPSS\_ERETRY.

## **Clients**

The RTM utility called rtmu.

## **8.3. RTM Library Routines**

### **8.3.1. rtm\_Init**

#### **Syntax**

```
signed32
rtm_Init (
    char                *ServerName,                /* IN */
    sec_authz_vector_t *ServerAuthzVector) ;        /* IN */
```

#### **Description**

This routine is called once by each server to register the RTM interface using **hpss\_RPCRegisterService** and to create a separate thread pool for this interface. It also initializes the RTM\_list mutex, copies the server authzvector to a known RTM global, opens the master catalog for log messages, and starts the thread that will delete entries from the delayed-delete lists (routine rtm\_delete\_thread - see pseudo code).

#### **Parameters**

<i>ServerName</i>	The server name.
<i>ServerAuthzVector</i>	The Server authorization vector.

#### **Return Values**

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

#### **Error Conditions**

HPSS_E_NOERROR	Initialization OK.
HPSS_EMUTEX_PROBLEM	Failed to initialize a mutex.

Errors as returned from **hpss\_RPCRegisterService**.

#### **See Also**

**rtm\_Shutdown**

#### **Clients**

Participating HPSS servers

#### **Notes**

None.



### 8.3.2. rtm\_Shutdown

#### Syntax

```
signed32
rtm_Shutdown (
void);
```

#### Description

This routine calls **hpss\_RPCUnregisterService** for the RTM interface.

#### Parameters

none

#### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

#### Error Conditions

Returns the error returned by **hpss\_RPCUnregisterService**.

#### See Also

**rtm\_Init**

#### Clients

Participating HPSS servers

#### Notes

None.

### 8.3.3. rtm\_ReqListInit

#### Syntax

```
#include "rtm_reqlist.h"
signed32
rtm_ReqListInit (
    unsigned32      ListName,                /* IN */
    char            *ServerDescName,         /* IN */
    unsigned32      ServerType,              /* IN */
    hpss_uuid_t     ServerId,                /* IN */
    rtm_reqlist_t   **ListPtrPtr);           /* OUT */
```

#### Description

This routine is an initialization routine to set up a request list. It locks the header list and initializes the header list itself the first time through. It mallocs space for a ReqList, and initializes the specific request list, including its protective mutex. It initializes the DeleteList associated with this request list. It then unlocks the header list.

#### Parameters

<i>ListName</i>	The request list to be initialized, e.g. RTM_CORE_DEFAULT_REQLIST.
<i>ServerDescName</i>	The Server Descriptive Name of the requesting server, e.g.CORE.
<i>ServerType</i>	The Server Type of the requesting server, e.g. RTM_CORE_SERVER.
<i>ServerId</i>	The Server Id of the requesting server.
<i>ListPtrPtr</i>	A pointer to the pointer to the request list that was initialized. This will be passed in subsequent calls to the rtm library routines.

### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

### Error Conditions

HPSS_E_NOERROR	Initialization OK.
HPSS_EINVAL	Initialization routine was already called.
HPSS_EMUTEX_PROBLEM	Failed to initialize mutex.
HPSS_ENOMEM	malloc failed.

### See Also

**rtm\_ReqListInsertEntry**

### Clients

Participating HPSS servers

### Notes

None.

## 8.3.4. rtm\_ReqListInsertEntry

### Syntax

```
#include "rtm_reqlist.h"
signed32
rtm_ReqListInsertEntry (
    rtm_reqlist_t          *ListPtr,          /* IN */
    hpss_reqid_t           ReqId,             /* IN */
    rtm_request_code_t     ReqCode,           /* IN */
    rtm_state_t            ReqState,          /* IN */
    rtm_req_entry_t        **EntryPtr);       /* OUT */
```

### Description

This routine is called by each participating HPSS server to insert a new request entry in the request list and return a pointer to the entry. It uses malloc() to create a new request list entry node. Multiple request entries per request id may be entered by a server if the request enters the server multiple times, or if the server wishes to separate the entries for any reason.

### Parameters

<i>ListPtr</i>	A pointer to the request list that was returned from the <b>rtm_ReqListInit</b> call.
<i>ReqId</i>	The Request ID of the current request.
<i>ReqCode</i>	The server-specific request code, e.g. RTM_CORE_CREATE_REQ.
<i>ReqState</i>	The request state, e.g. RTM_REQ_IN_PROGRESS.
<i>EntryPtr</i>	A pointer to the new request list entry pointer.

### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

### Error Conditions

HPSS_E_NOERROR	Initialization OK.
HPSS_EINVAL	Invalid <i>ListPtr</i> was passed in.
HPSS_EMUTEX_PROBLEM	Failed to lock or unlock a mutex
HPSS_ENOMEM	Malloc failed.

### See Also

**rtm\_ReqListDeleteEntry**

### Clients

Participating HPSS servers

### Notes

None.

## 8.3.5. rtm\_ReqListDeleteEntry

### Syntax

```
#include "rtm_reqlist.h"
signed32
rtm_ReqListDeleteEntry (
    rtm_reqlist_t          *ListPtr,          /* IN */
    hpss_reqid_t           ReqId,             /* IN */
    rtm_req_entry_t        *EntryPtr,         /* IN */
    unsigned32              Seconds);          /* IN */
```

### Description

This routine is called by each participating HPSS server to delete a request entry from the request list. It scans the request list looking for this entry. If *Seconds* is zero, it unlinks the entry, and frees the memory associated with the request entry and all its waitlist entries. If *Seconds* is non-zero, it puts the request entry pointer onto the DeleteList.

### Parameters

<i>ListPtr</i>	A pointer to the request list that was returned from the <b>rtm_ReqListInit</b> call.
<i>ReqId</i>	The Request ID of the current request.

<i>EntryPtr</i>	A pointer to the entry to delete.
<i>Seconds</i>	The number of seconds to delay before deleting this request entry.

### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

### Error Conditions

HPSS_E_NOERROR	Initialization OK.
HPSS_EINVAL	Invalid argument was passed in.
HPSS_EMUTEX_PROBLEM	Failed to lock or unlock a mutex
HPSS_ESYSTEM	Software error.

### See Also

**rtm\_ReqListInsertEntry**

### Clients

Participating HPSS servers

### Notes

None.

## 8.3.6. rtm\_ReqListUpdateEntry

### Syntax

```
#include "rtm_reqlist.h"
signed32
rtm_ReqListUpdateEntry (
    rtm_reqlist_t          *ListPtr,          /* IN */
    hpss_reqid_t           ReqId,             /* IN */
    rtm_req_entry_t        *EntryPtr,         /* IN */
    rtm_state_t            ReqState);         /* IN */
```

### Description

This routine is called by each participating HPSS server to update the state of an existing request entry.

### Parameters

<i>ListPtr</i>	A pointer to the request list that was returned from the <b>rtm_ReqListInit</b> call.
<i>ReqId</i>	The Request ID of the request entry to update.
<i>EntryPtr</i>	A pointer to the request entry to update.
<i>ReqState</i>	The new request state.

### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

### Error Conditions

HPSS_E_NOERROR	Initialization OK.
HPSS_EINVAL	<i>EntryPtr</i> argument is NULL.
HPSS_EMUTEX_PROBLEM	Failed to lock or unlock a mutex

#### See Also

**rtm\_ReqListInsertEntry**, **rtm\_ReqListDeleteEntry**

#### Clients

Participating HPSS servers

#### Notes

None.

### 8.3.7. rtm\_WaitListInsertEntry

#### Syntax

```
#include "rtm_reqlist.h"
signed32
rtm_WaitListInsertEntry (
    rtm_reqlist_t          *ListPtr,          /* IN */
    hpss_reqid_t           ReqId,             /* IN */
    rtm_state_t            ReqState,          /* IN */
    rtm_req_entry_t        *EntryPtr,         /* IN */
    rtm_serverspecific_t   *ServerSpecific,   /* IN */
    rtm_wait_reason_t      WaitReason,        /* IN */
    rtm_wait_resource_t     WaitResource,     /* IN */
    rtm_wait_entry_t       **WaitEntryPtr);  /* OUT */
```

#### Description

This routine is called by an HPSS server to insert a new waitlist entry into the request entry indicated by *EntryPtr*. It uses malloc() to create a new waitlist entry node, inserts this waitlist entry at the front of the request's waitlist, and returns a pointer to the waitlist entry.

#### Parameters

<i>ListPtr</i>	A pointer to the request list that was returned from the <b>rtm_ReqListInit</b> call.
<i>ReqId</i>	The Request ID of the request entry of interest.
<i>ReqState</i>	The current Request State of the request entry of interest, or zero for no change.
<i>EntryPtr</i>	A pointer to the request entry to add this waitlist entry to.
<i>Serverspecific</i>	The server-specific structure that pertains to this waitlist entry.
<i>WaitReason</i>	An enumerated list that explains why the server is in a wait state.
<i>WaitResource</i>	The WaitResource being waited on, e.g. a volume id.
<i>WaitEntryPtr</i>	A pointer to the new waitlist entry pointer.

#### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

#### Error Conditions

HPSS_E_NOERROR	Initialization OK.
HPSS_EMUTEX_PROBLEM	Failed to lock/unlock a mutex
HPSS_ENOMEM	Malloc failed.
HPSS_EINVAL	Invalid value was passed in.

#### See Also

**rtm\_ReqListInsertEntry**, **rtm\_ReqListDeleteEntry**, **rtm\_WaitListDeleteEntry**

#### Clients

Participating HPSS servers

#### Notes

None.

### 8.3.8. rtm\_WaitListDeleteEntry

#### Syntax

```
#include rtm_reqlist.h
signed32
rtm_WaitListDeleteEntry (
    rtm_reqlist_t          *ListPtr,          /* IN */
    hpss_reqid_t           ReqId,             /* IN */
    rtm_state_t            ReqState,          /* IN */
    rtm_req_entry_t        *EntryPtr,         /* IN */
    rtm_wait_entry_t       *WaitEntryPtr);    /* IN */
```

#### Description

This routine is called by an HPSS server to delete a waitlist entry from the given request entry. It locks the request list, scans the waitlist entries for the given request entry, and deletes the waitlist entry and frees the memory.

#### Parameters

<i>ListPtr</i>	A pointer to the request list that was returned from the <b>rtm_ReqListInit</b> call.
<i>ReqId</i>	The Request ID of the current request.
<i>ReqState</i>	The current Request State of the request entry of interest, or zero for no change.
<i>EntryPtr</i>	A pointer to the request entry from which to delete this waitlist entry.
<i>WaitEntryPtr</i>	A pointer to the waitlist entry to delete.

#### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

#### Error Conditions

HPSS_EINVAL	Invalid argument was passed in.
HPSS_ENOENT	Waitlist entry not found.
HPSS_E_NOERROR	Initialization OK.
HPSS_EMUTEX_PROBLEM	Failed to lock/unlock a mutex.

#### See Also

**rtm\_ReqListInsertEntry, rtm\_ReqListDeleteEntry, rtm\_WaitListInsertEntry**

#### Clients

Participating HPSS servers

#### Notes

None.

## 8.3.9. rtm\_WaitListUpdateEntry

#### Syntax

```
#include "rtm_reqlist.h"
signed32
rtm_WaitListUpdateEntry (
    rtm_reqlist_t          *ListPtr,          /* IN */
    hpss_reqid_t           ReqId,             /* IN */
    rtm_state_t            ReqState,          /* IN */
    rtm_req_entry_t        *EntryPtr,         /* IN */
    rtm_wait_entry_t       *WaitEntryPtr,     /* IN */
    rtm_serverspecific_t   *ServerSpecific,   /* IN */
    rtm_wait_reason_t       WaitReason,       /* IN */
    rtm_wait_resource_t     WaitResource);    /* IN */
```

#### Description

This routine is called by an HPSS server to update the waitlist entry indicated by *WaitEntryPtr*, which is a member of the waitlist in the request entry *EntryPtr*.

#### Parameters

<i>ListPtr</i>	A pointer to the request list that was returned from the <b>rtm_ReqListInit</b> call.
<i>ReqId</i>	The Request ID of the current request.
<i>ReqState</i>	The current Request State of the request entry of interest, or zero for no change.
<i>EntryPtr</i>	A pointer to the request entry containing this waitlist entry.
<i>WaitEntryPtr</i>	A pointer to the waitlist entry to update.
<i>Serverspecific</i>	The server-specific struct that pertains to this waitlist entry.
<i>WaitReason</i>	An enumerated integer that explains why the server is in a wait state.
<i>WaitResource</i>	The WaitResource being waited on, e.g. a volume id.

#### Return Values

Zero indicates success. A non-zero return indicates an error and is a code that defines the error.

#### **Error Conditions**

HPSS_E_NOERROR	Initialization OK.
HPSS_EINVAL	Invalid argument passed in.
HPSS_EMUTEX_PROBLEM	Failed to lock/unlock a mutex.

#### **See Also**

**rtm\_WaitListInsertEntry, rtm\_WaitListInsertEntry**

#### **Clients**

Participating HPSS servers

#### **Notes**

None.



## Appendix A. Glossary of Terms and Acronyms

<b>ACI</b>	Automatic Media Library Client Interface
<b>ACL</b>	Access Control List
<b>ACSLs</b>	Automated Cartridge System Library Software (Science Technology Corporation)
<b>ADIC</b>	Advanced Digital Information Corporation
<b>accounting</b>	The process of tracking system usage per user, possibly for the purposes of charging for that usage. Also, a log record message type used to log information to be used by the HPSS Accounting process. This message type is not currently used.
<b>AIX</b>	Advanced Interactive Executive. An operating system provided on many IBM machines.
<b>alarm</b>	A log record message type used to log high-level error conditions.
<b>AML</b>	Automated Media Library. A tape robot.
<b>AMS</b>	Archive Management Unit
<b>ANSI</b>	American National Standards Institute
<b>API</b>	Application Program Interface
<b>archive</b>	One or more interconnected storage systems of the same architecture.
<b>attribute</b>	When referring to a managed object, an attribute is one discrete piece of information, or set of related information, within that object.
<b>attribute change</b>	When referring to a managed object, an attribute change is the modification of an object attribute. This event may result in a notification being sent to SSM, if SSM is currently registered for that attribute.
<b>audit (security)</b>	An operation that produces lists of HPSS log messages whose record type is SECURITY. A security audit is used to provide a trail of security-relevant activity in HPSS.
<b>bar code</b>	An array of rectangular bars and spaces in a predetermined pattern which represent alphanumeric information in a machine readable format (e.g., UPC symbol)
<b>BFS</b>	HPSS Bitfile Service.

<b>bitfile</b>	A file stored in HPSS, represented as a logical string of bits unrestricted in size or internal structure. HPSS imposes a size limitation in 8-bit bytes based upon the maximum size in bytes that can be represented by a 64-bit unsigned integer.
<b>bitfile segment</b>	An internal metadata structure, not normally visible, used by the Core Server to map contiguous pieces of a bitfile to underlying storage.
<b>Bitfile Service</b>	Portion of the HPSS Core Server that provides a logical abstraction of bitfiles to its clients.
<b>BMUX</b>	Block Multiplexer Channel
<b>bytes between tape marks</b>	The number of data bytes that are written to a tape virtual volume before a tape mark is required on the physical media.
<b>CAP</b>	Cartridge Access Port
<b>cartridge</b>	A physical media container, such as a tape reel or cassette, capable of being mounted on and dismounted from a drive. A fixed disk is technically considered to be a cartridge because it meets this definition and can be logically mounted and dismounted.
<b>central log</b>	The main repository of logged messages from all HPSS servers enabled to send messages to the Log Daemon.
<b>class</b>	A type definition in Java. It defines a template on which objects with similar characteristics can be built, and includes variables and methods specific to the class.
<b>Class of Service</b>	A set of storage system characteristics used to group bitfiles with similar logical characteristics and performance requirements together. A Class of Service is supported by an underlying hierarchy of storage classes.
<b>cluster</b>	The unit of storage space allocation on HPSS disks. The smallest amount of disk space that can be allocated from a virtual volume is a cluster. The size of the cluster on any given disk volume is determined by the size of the smallest storage segment that will be allocated on the volume, and other factors.
<b>configuration</b>	The process of initializing or modifying various parameters affecting the behavior of an HPSS server or infrastructure service.
<b>COS</b>	Class of Service
<b>Core Server</b>	An HPSS server which manages the namespace and storage for an HPSS system. The Core Server manages the Name Space in which files are defined, the attributes of the files, and the storage media on which the files are stored. The Core Server is the central server of an HPSS system. Each storage sub-system uses exactly one Core Server.
<b>daemon</b>	A UNIX program that runs continuously in the background.

<b>DB2</b>	A relational database system, a product of IBM Corporation, used by HPSS to store and manage HPSS system metadata.
<b>debug</b>	A log record message type used to log lower-level error conditions.
<b>DEC</b>	Digital Equipment Corporation.
<b>delog</b>	The process of extraction, formatting, and outputting HPSS central log records.
<b>deregistration</b>	The process of disabling notification to SSM for a particular attribute change.
<b>descriptive name</b>	A human-readable name for an HPSS server.
<b>device</b>	A physical piece of hardware, usually associated with a drive, that is capable of reading or writing data.
<b>directory</b>	An HPSS object that can contain files, symbolic links, hard links, and other directories.
<b>dismount</b>	An operation in which a cartridge is either physically or logically removed from a device, rendering it unreadable and unwritable. In the case of tape cartridges, a dismount operation is a physical operation. In the case of a fixed disk unit, a dismount is a logical operation.
<b>DNS</b>	Domain Name Service
<b>DOE</b>	Department of Energy
<b>drive</b>	A physical piece of hardware capable of reading and/or writing mounted cartridges. The terms device and drive are often used interchangeably.
<b>EFS</b>	External File System
<b>ERA</b>	Extended Registry Attribute
<b>ESCON</b>	Enterprise System Connection
<b>event</b>	A log record message type used to log informational messages (e.g., subsystem starting, subsystem terminating).
<b>export</b>	\nAn operation in which a cartridge and its associated storage space are removed from the HPSS system Physical Volume Library. It may or may not include an eject, which is the removal of the cartridge from its Physical Volume Repository.
<b>FDDI</b>	Fiber Distributed Data Interface
<b>file</b>	An object than can be written to, read from, or both, with attributes including access permissions and type, as defined by POSIX (P1003.1-1990). HPSS supports only regular files.

<b>file family</b>	An attribute of an HPSS file that is used to group a set of files on a common set of tape virtual volumes.
<b>fileset</b>	A collection of related files that are organized into a single easily managed unit. A fileset is a disjoint directory tree that can be mounted in some other directory tree to make it accessible to users.
<b>fileset ID</b>	A 64-bit number that uniquely identifies a fileset.
<b>fileset name</b>	A name that uniquely identifies a fileset.
<b>file system ID</b>	A 32-bit number that uniquely identifies an aggregate.
<b>FTP</b>	File Transfer Protocol
<b>Gatekeeper</b>	An HPSS server that provides two main services: the ability to schedule the use of HPSS resources referred to as the Gatekeeping Service, and the ability to validate user accounts referred to as the Account Validation Service.
<b>Gatekeeping Service</b>	A registered interface in the Gatekeeper that provides a site the mechanism to create local policy on how to throttle or deny create, open and stage requests and which of these request types to monitor.
<b>Gatekeeping Site Interface</b>	The APIs of the gatekeeping site policy code.
<b>Gatekeeping Site Policy</b>	The gatekeeping shared library code written by the site to monitor and throttle create, open, and/or stage requests.
<b>GB</b>	Gigabyte ( $2^{30}$ )
<b>GECOS</b>	The comment field in a UNIX password entry that can contain general information about a user, such as office or phone number.
<b>GID</b>	Group Identifier
<b>GK</b>	Gatekeeper
<b>GSS</b>	Generic Security Service
<b>GUI</b>	Graphical User Interface
<b>HA</b>	High Availability
<b>HACMP</b>	High Availability Clustered Multi-Processing - A software package used to implement high availability systems.
<b>halt</b>	A forced shutdown of an HPSS server.
<b>HDM</b>	Shorthand for HPSS/DMAP.
<b>hierarchy</b>	See Storage Hierarchy.

<b>HIMF</b>	HPSS Interim Metadata Format
<b>HiPPI</b>	High Performance Parallel Interface
<b>HPSS</b>	High Performance Storage System
<b>HPSS-only fileset</b>	An HPSS fileset that is not linked to an external filesystem (e.g., XFS).
<b>IBM</b>	International Business Machines Corporation
<b>ID</b>	Identifier
<b>IEC</b>	International Electrotechnical Commission
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IETF</b>	Internet Engineering Task Force
<b>Imex</b>	Import/Export
<b>import</b>	An operation in which a cartridge and its associated storage space are made available to the HPSS system. An import requires that the cartridge has been physically introduced into a Physical Volume Repository (injected). Importing the cartridge makes it known to the Physical Volume Library.
<b>I/O</b>	Input/Output
<b>IOD/IOR</b>	Input/Output Descriptor / Input/Output Reply. Structures used to send control information about data movement requests in HPSS and about the success or failure of the requests.
<b>IP</b>	Internet Protocol
<b>IRIX</b>	SGI's implementation of UNIX
<b>junction</b>	A mount point for an HPSS fileset.
<b>KB</b>	Kilobyte (210)
<b>LAN</b>	Local Area Network
<b>LANL</b>	Los Alamos National Laboratory
<b>LARC</b>	Langley Research Center
<b>latency</b>	For tape media, the average time in seconds between the start of a read or write request and the time when the drive actually begins reading or writing the tape.
<b>LCU</b>	Library Control Unit
<b>LDAP</b>	Lightweight Directory Access Protocol

<b>LLNL</b>	Lawrence Livermore National Laboratory
<b>LMCP</b>	Library Manager Control Point
<b>LMU</b>	Library Management Unit
<b>local log</b>	An optional circular log maintained by a Log Client. The central log contains formatted messages from all enabled HPSS servers residing on the same node as the Log Client.
<b>Location Server</b>	An HPSS server that is used to help clients locate the appropriate Core Server and/or other HPSS server to use for a particular request.
<b>Log Client</b>	An HPSS server executing on each HPSS node that is responsible for sending log messages to the local log, to the Log Daemon for central logging, and to SSM to display messages in the Alarm and Event window.
<b>Log Daemon</b>	An HPSS server responsible for writing log messages to the central log.
<b>log record</b>	A record received and maintained in a central log by the HPSS Log Daemon.
<b>log record type</b>	An indicator of whether a message to be logged is an alarm, event, status, debug, request, security, or accounting record.
<b>logging service</b>	An HPSS infrastructure service consisting of a central Log Daemon, one or more Log Clients, and server-specific logging policies.
<b>LRU</b>	Least Recently Used
<b>LS</b>	Location Server
<b>LTO</b>	Linear Tape-Open. A half-inch open tape technology developed by IBM, HP and Seagate.
<b>MAC</b>	Mandatory Access Control
<b>managed object</b>	A programming data structure that represents an HPSS system resource. The resource can be monitored and controlled by operations on the managed object. Managed objects in HPSS are used to represent servers, drives, storage media, jobs, and other resources.
<b>MB</b>	Megabyte (220)
<b>metadata</b>	Control information about the data stored under HPSS, such as location, access times, permissions, and storage policies. Most HPSS metadata is stored in a DB2 relational database.
<b>method</b>	A Java function or subroutine
<b>migrate</b>	To copy file data from a level in the file's hierarchy onto the next lower level in the hierarchy.

<b>Migration/Purge Server</b>	An HPSS server responsible for supervising the placement of data in the storage hierarchies based upon site-defined migration and purge policies.
<b>MM</b>	Metadata Manager. A software library that provides a programming API to interface HPSS servers with the DB2 programming environment.
<b>mount</b>	An operation in which a cartridge is either physically or logically made readable and/or writable on a drive. In the case of tape cartridges, a mount operation is a physical operation. In the case of a fixed disk unit, a mount is a logical operation.
<b>mount point</b>	A place where a fileset is mounted in the XFS and/or HPSS namespaces.
<b>Mover</b>	An HPSS server that provides control of storage devices and data transfers within HPSS.
<b>MPS</b>	Migration/Purge Server
<b>MRA</b>	Media Recovery Archive
<b>MSSRM</b>	Mass Storage System Reference Model
<b>MVR</b>	Mover
<b>NASA</b>	National Aeronautics and Space Administration
<b>Name Service</b>	The portion of the Core Server that provides a mapping between names and machine oriented identifiers. In addition, the Name Service performs access verification and provides the Portable Operating System Interface (POSIX).
<b>name space</b>	The set of name-object pairs managed by the HPSS Core Server.
<b>NERSC</b>	National Energy Research Supercomputer Center
<b>NLS</b>	National Language Support
<b>notification</b>	A notice from one server to another about a noteworthy occurrence. HPSS notifications include notices sent from other servers to SSM of changes in managed object attributes, changes in tape mount information, and log messages that are alarm, event, and status log record message types.
<b>NS</b>	HPSS Name Service
<b>NSL</b>	National Storage Laboratory
<b>object</b>	See Managed Object
<b>ONC</b>	Open Network Computing
<b>ORNL</b>	Oak Ridge National Laboratory
<b>OSF</b>	Open Software Foundation
<b>OS/2</b>	Operating System (multi-tasking, single user) used on the AMU controller PC

<b>PB</b>	Petabyte (2 <sup>50</sup> )
<b>PFTP</b>	Parallel File Transfer Protocol
<b>physical volume</b>	An HPSS object managed jointly by the Core Server and the Physical Volume Library that represents the portion of a virtual volume. A virtual volume may be composed of one or more physical volumes, but a physical volume may contain data from no more than one virtual volume.
<b>Physical Volume Library</b>	An HPSS server that manages mounts and dismounts of HPSS physical volumes.
<b>Physical Volume Repository</b>	An HPSS server that manages the robotic agent responsible for mounting and dismounting cartridges or interfaces with the human agent responsible for mounting and dismounting cartridges.
<b>PIO</b>	Parallel I/O
<b>PIOFS</b>	Parallel I/O File System
<b>POSIX</b>	Portable Operating System Interface (for computer environments)
<b>purge</b>	Deletion of file data from a level in the file's hierarchy after the data has been duplicated at lower levels in the hierarchy and is no longer needed at the deletion level.
<b>purge lock</b>	A lock applied to a bitfile which prohibits the bitfile from being purged.
<b>PV</b>	Physical Volume
<b>PVL</b>	Physical Volume Library
<b>PVM</b>	Physical Volume Manager
<b>PVR</b>	Physical Volume Repository
<b>RAID</b>	Redundant Array of Independent Disks
<b>RAIT</b>	Redundant Array of Independent Tapes
<b>RAM</b>	Random Access Memory
<b>reclaim</b>	The act of making empty tape virtual volumes available for reuse. Reclaimed tape virtual volumes are assigned a new Virtual Volume ID, but retain the rest of their previous characteristics.
<b>registration</b>	The process by which SSM requests notification of changes to specified attributes of a managed object.
<b>reinitialization</b>	An HPSS SSM administrative operation that directs an HPSS server to reread its latest configuration information, and to change its operating parameters to match that configuration, without going through a server shutdown and restart.



<b>repack</b>	The act of moving data from a virtual volume onto another virtual volume with the same characteristics with the intention of removing all data from the source virtual volume.
<b>request</b>	A log record message type used to log some action being performed by an HPSS server on behalf of a client.
<b>RISC</b>	Reduced Instruction Set Computer/Cycles
<b>RMS</b>	Removable Media Service
<b>RPC</b>	Remote Procedure Call
<b>SCSI</b>	Small Computer Systems Interface
<b>security</b>	A log record message type used to log security related events (e.g., authorization failures).
<b>SGI</b>	Silicon Graphics
<b>shelf tape</b>	A cartridge which has been physically removed from a tape library but whose file metadata still resides in HPSS.
<b>shutdown</b>	An HPSS SSM administrative operation that causes a server to stop its execution gracefully.
<b>sink</b>	The set of destinations to which data is sent during a data transfer (e.g., disk devices, memory buffers, network addresses).
<b>SMC</b>	SCSI Medium Changer
<b>SMIT</b>	System Management Interface Tool
<b>SNL</b>	Sandia National Laboratories
<b>SOID</b>	Storage Object ID. An internal HPSS storage object identifier that uniquely identifies a storage resource. The SOID contains an unique identifier for the object, and an unique identifier for the server that manages the object.
<b>source</b>	The set of origins from which data is received during a data transfer (e.g., disk devices, memory buffers, network addresses).
<b>SP</b>	Scalable Processor
<b>SS</b>	HPSS Storage Service
<b>SSA</b>	Serial Storage Architecture
<b>SSM</b>	Storage System Management

<b>SSM session</b>	The environment in which an SSM user interacts with the SSM System Manager to monitor and control HPSS. This environment may be the graphical user interface provided by the hpssgui program, or the command line user interface provided by the hpssadm program.
<b>stage</b>	To copy file data from a level in the file's hierarchy onto the top level in the hierarchy.
<b>start-up</b>	An HPSS SSM administrative operation that causes a server to begin execution.
<b>status</b>	A log record message type used to log processing results. This message type is being used to report status from the HPSS Accounting process.
<b>STK</b>	Storage Technology Corporation
<b>storage class</b>	An HPSS object used to group storage media together to provide storage for HPSS data with specific characteristics. The characteristics are both physical and logical.
<b>storage hierarchy</b>	An ordered collection of storage classes. The hierarchy consists of a fixed number of storage levels numbered from level 1 to the number of levels in the hierarchy, with the maximum level being limited to 5 by HPSS. Each level is associated with a specific storage class. Migration and stage commands result in data being copied between different storage levels in the hierarchy. Each Class of Service has an associated hierarchy.
<b>storage level</b>	The relative position of a single storage class in a storage hierarchy. For example, if a storage class is at the top of a hierarchy, the storage level is 1.
<b>storage map</b>	An HPSS object managed by the Core Server to keep track of allocated storage space.
<b>storage segment</b>	An HPSS object managed by the Core Server to provide abstract storage for a bitfile or parts of a bitfile.
<b>Storage Service</b>	The portion of the Core Server which provides control over a hierarchy of virtual and physical storage resources.
<b>storage subsystem</b>	A portion of the HPSS namespace that is managed by an independent Core Server and (optionally) Migration/Purge Server.
<b>Storage System Management</b>	An HPSS component that provides monitoring and control of HPSS via a windowed operator interface or command line interface.
<b>stripe length</b>	The number of bytes that must be written to span all the physical storage media (physical volumes) that are grouped together to form the logical storage media (virtual volume). The stripe length equals the virtual volume block size multiplied by the number of physical volumes in the stripe group (i.e., stripe width).

<b>stripe length</b>	The number of bytes that must be written to span all the physical storage media (physical volumes) that are grouped together to form the logical storage media (virtual volume). The stripe length equals the virtual volume block size multiplied by the number of physical volumes in the stripe group (i.e., stripe width).
<b>stripe width</b>	The number of physical volumes grouped together to represent a virtual volume.
<b>System Manager</b>	The Storage System Management (SSM) server. It communicates with all other HPSS components requiring monitoring or control. It also communicates with the SSM graphical user interface (hpssgui) and command line interface (hpssadm).
<b>TB</b>	Terabyte ( $2^{40}$ )
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>trace</b>	A log record message type used to record entry/exit processing paths through HPSS server software.
<b>transaction</b>	<p>A programming construct that enables multiple data operations to possess the following properties:</p> <ul style="list-style-type: none"> <li>All operations commit or abort/roll-back together such that they form a single unit of work.</li> <li>All data modified as part of the same transaction are guaranteed to maintain a consistent state whether the transaction is aborted or committed.</li> <li>Data modified from one transaction are isolated from other transactions until the transaction is either committed or aborted.</li> <li>Once the transaction commits, all changes to data are guaranteed to be permanent.</li> </ul>
<b>TTY</b>	Teletypewriter
<b>UDP</b>	User Datagram Protocol
<b>UID</b>	User Identifier
<b>UPC</b>	Universal Product Code
<b>UUID</b>	Universal Unique Identifier
<b>virtual volume</b>	An HPSS object managed by the Core Server that is used to represent logical media. A virtual volume is made up of a group of physical storage media (a stripe group of physical volumes).
<b>virtual volume block size</b>	The size of the block of data bytes that is written to each physical volume of a striped virtual volume before switching to the next physical volume.
<b>VV</b>	Virtual Volume

<b>XDSM</b>	The Open Group's Data Storage Management standard. It defines APIs that use events to notify Data Management applications about operations on files.
<b>XFS</b>	A file system created by SGI available as open source for the Linux operating system.



## Appendix B. References

1. *3580 Ultrium Tape Drive Setup, Operator and Service Guide* GA32-0415-00
2. *3584 UltraScalable Tape Library Planning and Operator Guide* GA32-0408-01
3. *3584 UltraScalable Tape Library SCSI Reference* WB1108-00
4. *AIX Performance Tuning Guide*
5. *Data Storage Management (XDSM) API*, ISBN 1-85912-190-X
6. *HACMP for AIX, Version 4.4: Concepts and Facilities*
7. *HACMP for AIX, Version 4.4: Planning Guide*
8. *HACMP for AIX, Version 4.4: Installation Guide*
9. *HACMP for AIX, Version 4.4: Administration Guide*
10. *HACMP for AIX, Version 4.4: Troubleshooting Guide*
11. *HPSS Error Messages Reference Manual*, July 2008, Release 6.2.
12. *HPSS Programmer's Reference* , July 2008, Release 6.2.
13. *HPSS Programmer's Reference – I/O Supplement* , July 2008, Release 6.2.
14. *HPSS User's Guide*, July 2008, Release 6.2.
15. *IBM 3494 Tape Library Dataserver Operator's Guide*, GA32-0280-02
16. *IBM AIX Version 4.3 Installation Guide*, SC23-4112-01
17. *IBM SCSI Device Drivers: Installation and User's Guide*, GC35-0154-01
18. *IBM Ultrium Device Drivers Installation and User's Guide* GA32-0430-00.1
19. *IBM Ultrium Device Drivers Programming Reference* WB1304-01
20. *Interfacing Guide DAS*, Order no. DOC F00 011
21. *Installing, Managing, and Using the IBM AIX Parallel I/O File System*, SH34-6065-02
22. *Parallel and ESCON Channel Tape Attachment/6000 Installation and User's Guide*, GA32-0311-02
23. *Platform Notes: The hme FastEthernet Device Driver* 805-4449
24. *POSIX 1003.1-1990 Tar Standard*
25. *Reference Guide AMU*, Order no. DOC E00 005
26. *STK Automated Cartridge System Library Software (ACSLs) System Administrator's Guide*,

27. ***STK Automated Cartridge System Library Software Programmer's Guide***, PN 16718
28. J. Steiner, C. Neuman, and J. Schiller, "***Kerberos: An Authentication Service for Open Network Systems***," USENIX 1988 Winter Conference Proceedings (1988).
29. R.W. Watson and R.A. Coyne, "***The Parallel I/O Architecture of the High-Performance Storage System (HPSS)***," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
30. T.W. Tyler and D.S. Fisher, "***Using Distributed OLTP Technology in a High-Performance Storage System***," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
31. J.K. Deutsch and M.R. Gary, "***Physical Volume Library Deadlock Avoidance in a Striped Media Environment***," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
32. R. Grossman, X. Qin, W. Xu, H. Hulen, and T. Tyler, "***An Architecture for a Scalable, High-Performance Digital Library***," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
33. S. Louis and R.D. Burris, "***Management Issues for High-Performance Storage Systems***," from the 1995 IEEE MSS Symposium, courtesy of the IEEE Computer Society Press.
34. D. Fisher, J. Sobolewski, and T. Tyler, "***Distributed Metadata Management in the High Performance Storage System***," from the 1st IEEE Metadata Conference, April 16-18, 1996.

## Appendix C. Developer Acknowledgments

HPSS is a product of a government-industry collaboration. The project approach is based on the premise that no single company, government laboratory, or research organization has the ability to confront all of the system-level issues that must be resolved for significant advancement in high-performance storage system technology.

HPSS development was performed jointly by IBM Worldwide Government Industry, Lawrence Berkeley National Laboratory, Lawrence Livermore National Laboratory, Los Alamos National Laboratory, NASA Langley Research Center, Oak Ridge National Laboratory, and Sandia National Laboratories.

We would like to acknowledge Argonne National Laboratory, the National Center for Atmospheric Research, and Pacific Northwest Laboratory for their help with initial requirements reviews.

We also wish to acknowledge Cornell Information Technologies of Cornell University for providing assistance with naming service and transaction management evaluations and for joint developments of the Name Service.

We also wish to acknowledge the many discussions, design ideas, implementation and operation experiences we have shared with colleagues at the National Storage Laboratory, the IEEE Mass Storage Systems and Technology Technical Committee, the IEEE Storage System Standards Working Group, and the storage community at large.

We also wish to acknowledge the Cornell Theory Center and the Maui High Performance Computer Center for providing a test bed for the initial HPSS release.

Finally, we wish to acknowledge CEA-DAM (**Commissariat à l'énergie Atomique - Centre d'études Bruyres-le-Château**) for providing assistance with development of NFS V3 protocol support.



